z/OS
2.4

MVS Programming: Authorized Assembler Services Reference, Volume 2 (EDT-IXG)

SA23-1373-40

Note

Before using this information and the product it supports, read the information in “Notices” on page 1463.

This edition applies to Version 2 Release 4 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2021-06-24

Contents

Figures

Tables

About this document

This document describes the authorized services that the MVS™ operating system provides; that is, services available only to authorized programs. An authorized program must meet one or more of the following requirements:

- Running in supervisor state
- Running under PSW key 0-7
- Running with APF-authorization.

Some of the services included in this document are not authorized, but are included because they are of greater interest to the system programmer than to the general applications programmer. The functions of these services are of such a nature that their use should be limited to programmers who write authorized programs. Services are also included if they have one or more authorized parameters — parameters available only to authorized programs.

Programmers using assembler language can use the macros described in this document to invoke the system services that they need. This document includes the detailed information — such as the function, syntax, and parameters — needed to code the macros.

This document is divided into four volumes. Volumes 1 through 4 present the macro descriptions in alphabetic order.

Who should use this document

This document is for the programmer who is using assembler language to code a system program. A system program is usually one that runs in supervisor state or runs with PSW key 0-7 or runs with APF authorization.

The document assumes a knowledge of the computer, as described in *Principles of Operation*, as well as an in-depth knowledge of assembler language programming.

System macros require High Level Assembler. For more information about assembler language programming, see High Level Assembler and Toolkit Feature in IBM Documentation (www.ibm.com/docs/en/hla-and-tf/1.6).

Using this information also requires you to be familiar with the operating system and the services that programs running under it can invoke.

How to use this document

This document is one of the set of programming documents for MVS. This set describes how to write programs in assembler language or high-level languages, such as C, FORTRAN, and COBOL. For more information about the content of this set of documents, see *z/OS Information Roadmap*.

z/OS information

This information explains how z/OS references information in other documents and on the web.

When possible, this information uses cross document links that go directly to the topic in reference 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*.

To find the complete z/OS® library, go to IBM Documentation (www.ibm.com/docs/en/zos).

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page lvii.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS MVS Authorized Assembler Services Reference EDT-IXG, SA23-1373-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority 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

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Summary of changes for z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG for Version 2 Release 4 (V2R4)

The following content is new, changed, or no longer included in V2R4.

New

The following new information is added in this publication:

June 2021 refresh

- The DSNTEST={NO|YES} option is added to the IEATDUMP service with APAR OA60029.

September 2020 refresh

- The REQUEST=CHANGEATTRIBUTE option is added to the IARV64 service via APAR OA58289. See REQUEST=CHANGEATTRIBUTE option of IARV64.

June 2020 refresh

- Clarification about the use of the COND parameter has been added in Chapter 34, “IARV64 — 64–bit virtual storage allocation,” on page 369.

Prior to June 2020 refresh

- The IFAWIC service has been added in Chapter 94, “IFAWIC — IBM z/OS Workload Interaction Correlator,” on page 953.
- The SENSITIVE parameter has been added in Chapter 29, “IARCP64 — 64-bit cell pool services,” on page 309, Chapter 31, “IARST64 — 64-bit storage services,” on page 333, “REQUEST=GETSTOR option of IARV64” on page 370, “REQUEST=GETSHARED option of IARV64” on page 387, and “REQUEST=GETCOMMON option of IARV64” on page 395 (APAR OA57633).
- z/OS sysplex events are added for ENFREQ. See ENF event codes and meanings .
- New return and reason code for Chapter 143, “IXGBRWSE — Browse/read a log stream,” on page 1351 is added for system logger enhancement to support security options for log stream unique access.
- The INORIGIN parameter is added to the IARV64 GETSTOR service. See “REQUEST=GETSTOR option of IARV64” on page 370.
- The ASYNCH parameter is added to the IEAARR service. See Chapter 37, “IEAARR — Establish an associated recovery routine (ARR),” on page 509.

Changed

The following information is changed in this publication:

June 2021

- For APAR OA60272, the ENFREQ service, ENF event code 36 is updated. See ENF event codes and meanings.
- For the ENQ service, the description of the RET=USE parameter has been updated. See Parameters.

- For the ISGENQ service, the description of the CONTENTIONACT=FAIL parameter has been updated. See “Parameters” on page 1210.

January 2021 refresh

- The syntax diagram for the GETMAIN service is updated to clarify the use of the LOC parameter. See Chapter 19, “GETMAIN — Allocate virtual storage,” on page 209.
- Information about the IEAFP service has been refreshed. See Chapter 38, “IEAFP — Floating point services,” on page 515.

October 2020 refresh

- For APAR OA57908, the INORIGIN parameter is updated in the IARV64 GETSTOR service. See “REQUEST=GETSTOR option of IARV64” on page 370.

September 2020 refresh

- The description of the CONVERT parameter has been updated in “REQUEST=CHANGEGUARD option of IARV64” on page 418 via APAR OA58289.

June 2020 refresh

- Chapter 112, “IOSODS — IOS offline device service,” on page 1095 parameters are updated in support of APAR OA56125.

Prior to June 2020 refresh

- Chapter 143, “IXGBRWSE — Browse/read a log stream,” on page 1351 and Chapter 144, “IXGCONN — Connect/disconnect to log stream,” on page 1399 are updated for system logger enhancement to support single-system scope Couple Data Set types (LOGRY and LOGRZ) for GDPS K-system environments.
- Return and reason code for Chapter 143, “IXGBRWSE — Browse/read a log stream,” on page 1351 is updated for system logger enhancement to support security options for log stream unique access.
- Chapter 144, “IXGCONN — Connect/disconnect to log stream,” on page 1399 is updated for system logger enhancement to support security options for log stream unique access.
- The ESTAEX macro has been updated. The SPIEOVERRIDE parameter no longer requires authorization. For more information, see “Environment” on page 102.
- In support of APAR OA55850, the IEFSJSYM macro SYMBAREA= parameter, return and reason codes, and the example are all updated. For more information, see “,SYMBAREA=xsymbarea” on page 873, Table 84 on page 874, and “Example” on page 875.
- The ORIGIN= parameter on the IARV64 REQUEST=GETSTORE is updated to describe the contents of the address returned to the caller when MEMLIMIT is exceed and if there is a RETCODE=8, see “,ORIGIN=origin” on page 384 for more information.

Deleted

The following information has been deleted from this publication:

June 2020 refresh

- The IARST64 service incorrectly included information about the EXECUTABLE parameter. The EXECUTABLE parameter has been removed from Chapter 31, “IARST64 — 64-bit storage services,” on page 333.

Summary of changes for z/OS Version 2 Release 3

The following information has been added, changed, or deleted in z/OS Version 2 Release 3 (V2R3). The most recent updates are listed at the top of each section.

New

- APAR OA52248 - The INTASKTERMOK parameter has been added to Chapter 44, “IEAMSXMP — Safe cross-memory post,” on page 559.
- Return code 06 has been added to Chapter 100, “IOSCHPD — IOS CHPID description service,” on page 1021.
- APAR OA53640 - Event code 19 has been added to ENF event codes and meanings.
- New macro Chapter 17, “FXECNTRL — Maintain Function Exploitation and Enablement,” on page 183.
- The USE2GTO32G and USE2GTO64G parameters have been added in “REQUEST=GETSTOR option of IARV64” on page 370.
- The ECMXAREA and ECMXLEN parameters have been added in Chapter 101, “IOSCMB — Locate the channel measurement block (CMB),” on page 1029.
- The OUTPUT_VERSION parameter has been added in Chapter 104, “IOSCUINF — Control unit information service,” on page 1043.
- The BYPHPALIAS parameter has been added in Chapter 112, “IOSODS — IOS offline device service,” on page 1095.
- The REXX and SCHSETSPEC parameters have been added in the “Description” on page 1121 of Chapter 115, “IOSSPOF — Check for single points of failure,” on page 1121.
- Added new EXECUTABLE parameter for API IARV64 to “REQUEST=GETSTOR option of IARV64” on page 370.
- New callable services IARBRVKR and IARBRVKA added. See Chapter 27, “IARBRVKA — Verify virtual storage access,” on page 305 and Chapter 28, “IARBRVKR — Replacement for the TPROT instruction,” on page 307.

Changed

- Information was added on when to use GRS RNL=NO. For more information see Parameters in Chapter 4, “ENQ — Request control of a serially reusable resource,” on page 71 and Parameters in Chapter 126, “ISGENQ — Global resource serialization ENQ service,” on page 1205.
- APAR OA50653 - Event code 33 has been updated in ENF event codes and meanings.
- Changes have been made to the MODE parameter and return code 04 in Chapter 145, “IXGDELET — Deleting log data from a log stream,” on page 1425 and Chapter 146, “IXGWRITE — Write log data to a log stream,” on page 1441.

Deleted

- Information about interruption type 17 has been removed from Chapter 5, “ESPIE — Extended SPIE,” on page 89.
- Removed last bullet point EUT from IESFP START in Chapter 38, “IEAFP — Floating point services,” on page 515.
- Removed note under VSA and NUMPAGES in “REQUEST=PAGEFIX option of IARV64” on page 456 and “REQUEST=PAGEUNFIX option of IARV64” on page 472.

Summary of changes for z/OS Version 2 Release 2 (V2R2), as updated December, 2015

The following changes are made for z/OS Version 2 Release 2 (V2R2), as updated December, 2015. In this revision, all technical changes for z/OS V2R2 are indicated by a vertical line to the left of the change.

New

- IARV64 has a new SADMP parameter for REQUEST=GETSTOR, REQUEST=GETSHARED, and REQUEST=GETCOMMON.
- The following callable services have been added:
  - Chapter 60, “IEAVPME2 — Pause multiple elements service,” on page 679
  - Chapter 74, “IEA4PME2 — 64-bit pause multiple elements service,” on page 759 .
- IXGCONN and IXGWRITE have a new return code 08, reason code xxxx084E.

Changed

- IARCP64 REQUEST=FREE now accepts the INPUT_CPID parameter.
- Restrictions are updated for the following callable services:
  - Chapter 56, “IEAVAPE — Allocate_Pause_Element,” on page 659
  - Chapter 70, “IEA4APE — Allocate_Pause_Element,” on page 739
  - Chapter 57, “IEAVAPE2 — Allocate_Pause_Element,” on page 663
  - Chapter 71, “IEA4APE2 — Allocate_Pause_Element,” on page 743.

Chapter 1. Using the services

Macros and callable services are programming interfaces that application programs can use to access MVS system services. This chapter provides general information and guidelines about how to use the macros and callable services accurately and efficiently. For more specific and detailed information about coding a particular macro or callable service, see the individual service description in this information.

Some of the topics covered in this chapter apply only to macros, some apply only to callable services, and some apply to both. This chapter uses the word "services" when referring to information that applies to both service types. When information applies only to one type or the other, the particular service type is specified.

Note: z/OS macros do not code to restrictions that are imposed by the COMPAT(CASE) HLASM option or its abbreviation CPAT(CASE). Therefore, you cannot rely on using COMPAT(CASE) if you use z/OS macros.

The following table lists the topics covered in this chapter and whether the topic applies to macros, callable services, or both:

Topic	Service Type
Compatibility of MVS macros	Macros
Addressing mode (AMODE)	Both
Address space control (ASC) mode	Both
ALET qualification	Both
User parameters	Macros
Telling the system about the execution environment	Macros
Specifying a macro version number	Macros
Register use	Both
Handling return codes and reason codes	Both
Handling program errors	Both
Handling environmental and system errors	Both
Using X-macros	Macros
Macro forms	Macros
Coding the macros	Macros
Coding the callable services	Callable Services
Including equate (EQU) statements	Callable Services
Link-editing linkage-assist routines	Callable Services
Service summary	Both

Compatibility of MVS macros

When IBM introduces a new version or a new release of an existing version, the new version or release supports all MVS macros from previous versions and releases. Programs assembled on an earlier level of MVS that issue macros will run on later levels of MVS.

In most cases, the reverse is also true. When you assemble programs that issue macros on a particular version and release of MVS, those programs can run on earlier versions and releases of MVS, provided you request only those functions that are supported by the earlier version and release. This is useful for

installations that write applications that might be assembled on one level of MVS, but run on a different level.

As MVS supports new architectures, addressability changes. To take best advantage of the new architectures, some macros have more than one possible expansion. You are required to have the macro expand according to the environment in which the program runs. This topic is described in this introductory information.

The problem of compatibility is not the same as selecting a macro version through the PLISTVER parameter to ensure the correct parameter list size for a macro. For selecting a parameter list version number, see Specifying a macro version number.

Addressing mode (AMODE)

A program can run in addressing mode (AMODE) 24, 31, or 64. A program that executes in AMODE 24 or 31can invoke most of the services described in this information. A program that executes in AMODE 64 has a smaller group of services that it can invoke.

In general:

- A program running in AMODE 24 cannot pass parameters or parameter addresses that are higher than 16 megabytes. However, there are exceptions. For example, a program running in AMODE 24 can:
  - Free storage above 16 megabytes using the FREEMAIN macro
  - Allocate storage above 16 megabytes using the GETMAIN macro
  - Use cell pool services for cell pools located in storage above 16 megabytes using the CPOOL macro
  - Use page services for storage locations above 16 megabytes using the PGSER macro
- A program is allowed to call a service from AMODE 64 only if the documentation for the service indicates that it supports AMODE 64.
- A program is allowed to call a service from RMODE 64 only if the documentation for the service indicates that it supports RMODE 64.
- A program running in AMODE 64 should not call a service with data, parameters, or parameter addresses that are higher than 2 gigabytes, unless the individual service description indicates that it is allowed.
- If a program running in AMODE 31 or 64 issues a service, parameters and parameter addresses can be above or below 16 megabytes, unless otherwise stated in the individual service description.

Some macros can generate code that is appropriate for programs in either AMODE 64 or 24 or 31. These macros check a global symbol set by the SYSSTATE macro. See Telling the system about the execution environment for more information.

When you call a callable service in AMODE 24 or 31, you must pass 31-bit addresses to the system service regardless of what addressing mode your program is running in. If your program is running in AMODE 24 and you use a callable service, you must set the high-order byte of parameter addresses to zeros.

You can invoke the following services in 64-bit addressing mode, subject to the “SVC or PC” restrictions mentioned later in this topic, but you cannot pass parameters and parameter addresses above 2 gigabytes: ABEND, ATTACHX, CALLDISP, CHAP, CSVQUERY, DELETE, DEQ, DETACH, DOM, DSPSERV, DYNALLOC, ENQ, ESPIE, ESTAEX, EXCP, FREEMAIN, GETMAIN, GTRACE, IARVSERV, IDENTIFY, IEAARR, LINKX, LOAD, MODESET, PGSER, POST, RESERVE, SDUMPX, SETRP, STAX, STIMER, STIMERM, STORAGE, SYNCHX, TIME, TIMEUSED, TTIMER, VRADATA, WAIT, WTO, WTOR, and XCTL.

There are many services that support AMODE 64 and parameter addresses above 2 gigabytes. Examples are IRAV64, IARST64, and ISGENQ. For details on the supported addressing mode and parameter address ranges for any specific service, see the following books:

- *z/OS MVS Programming: Assembler Services Reference ABE-HSP*
- *z/OS MVS Programming: Assembler Services Reference IAR-XCT*

- *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*
- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*
- *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*
- *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*
- *z/OS MVS Programming: Sysplex Services Reference*

Before invoking a service in AMODE 64, you must inform system macros, by specifying SYSSTATE AMODE64=YES. You can invoke only those options that result in calling the system by an SVC or PC in AMODE 64. You cannot invoke any option that results in calling the system by a branch-entry in AMODE 64.

Unless explicitly stated otherwise, assume that a given service cannot be invoked in AMODE 64 and cannot accept data, parameters, or parameter addresses above 2 gigabytes. Such an explicit statement would include a specific reference to AMODE 64 in a macro's environment section and additional information would mention that data, parameters, and parameter addresses could be above 2 gigabytes. By contrast, an AMODE specification of "Any" means that the macro can be invoked in either AMODE 24 or 31; it does not mean that the macro can be invoked in AMODE 64.

For information about AMODE 64 and the 64-bit GPR, see *z/OS MVS Programming: Extended Addressability Guide*.

Address space control (ASC) mode

A program can run in either primary ASC mode or access register (AR) ASC mode. In primary mode, the processor uses the contents of general purpose registers (GPRs) to resolve an address to a specific location. In AR mode, the processor uses the contents of ARs as well as the contents of GPRs to resolve an address to a specific location. See *z/OS MVS Programming: Assembler Services Guide* for more detailed information about AR mode.

Some macros can generate code that is appropriate for programs in either primary mode or AR mode. These macros check a global symbol set by the SYSSTATE macro. See Telling the system about the execution environment for more information. Table 1 lists the macros that check the global symbol.

Some services can generate code that is appropriate for programs in primary mode only. If you write a program in AR mode that invokes one or more services, check the description in this information for each service your program issues. Unless the description indicates that a service supports callers in AR mode, the service *does not* support callers in AR mode. In this case, use the SAC instruction to change the ASC mode of your program and issue the service in primary mode.

Whether the caller is in primary or AR ASC mode, the system uses ARs 0-1 and 14-15 as work registers across any service call.

ALET qualification

The address space where you can place parameters varies with the individual service:

- You can place parameters in the primary address space in all service.
- You must place parameters in the primary address space in some services.
- You can place parameters in any address space in some services.

To identify where you can locate parameters in a service, read the individual service description.

Programs in AR mode that pass parameters must use an access register and the corresponding general purpose register together (for example, access register 1 and general purpose register 1) to identify where the parameters are located. The access register must contain an access list entry token (ALET) that identifies the address space where the parameters reside. The general purpose register must identify the location of the parameters within the address space.

The only ALETs that MVS services typically accept are:

- Zero (0), which specifies that the parameters are in the caller's primary address space

- An ALET for a public entry on the caller's dispatchable unit access list (DU-AL)
- An ALET for a common area data space (CADS)

MVS services do not accept the following ALETs, and you cannot attempt to pass them to a service:

- One (1), which signifies that the parameters are in the caller's secondary address space
- An ALET that is on the caller's primary address space access list (PASN-AL) that does not represent a CADS
- An ALET for a private entry on the PASN-AL or the DU-AL

Throughout, this information uses the term **AR/GPR** *n* to mean an access register and its corresponding general purpose register. For example, to identify access register 1 and general purpose register 1, this information uses **AR/GPR 1**.

User parameters

Some macros that you can issue in AR mode include control parameters, user parameters, or both. Control parameters refer to the macro parameter list, and the parameters whose addresses are in the parameter list. Control parameters control the operation of the macro itself. User parameters are parameters that a user provides to be passed through to a user routine. For example, the PARAM parameter on the ATTACHX macro defines user parameters. The ATTACHX macro passes these parameters to the routine that it attaches. All other parameters on the ATTACHX macro are control parameters that control the operation of the ATTACHX macro.

Note:

1. User parameters are sometimes referred to as problem program parameters.
2. Control parameters are sometimes referred to as system parameters or control program parameters.

The macros shown in Table 1 allow a caller in AR mode to pass information in the form of a parameter list (or parameter lists) to another routine. This table identifies the parameter that receives the ALET-qualified address of the parameter list and tells you where the target routine finds the ALET-qualified address.

Table 1. Passing User Parameters in AR Mode

Macro	Parameter	Location of User Parameter List Address
ATTACH/ATTACHX	PARAM,VL=1	AR/GPR 1 contains the address of a list of addresses. When either • a 4-bytes-per-entry parameter list or • an 8-bytes-per-entry parameter list with PLIST8ARALETS=YES is being used, this list also contains the ALETs associated with those addresses. (See Figure 1 for the format of the 4-bytes-per-entry parameter list when it contains ALETs.)
ESTAEX	PARAM	SDWAPARM contains the address of an 8-byte area, which contains the address and ALET of the parameter list.

When an AR mode caller who is using a 4-bytes-per-entry parameter list passes ALET-qualified addresses to the called program through PARAM,VL=1 on the ATTACH/ATTACHX macro, the system builds a list formatted as shown in Figure 1. The addresses passed to the called program are at the beginning of the list, and their associated ALETs follow the addresses. The last address in the list has the high-order bit on to indicate the end of the list. For example, Figure 1 shows the format of a list where an AR mode issuer of ATTACHX who is using a 4-bytes-per-entry parameter list has coded the PARAM parameter as follows:

```
PARAM=(A,B,C),VL=1
```

When an AR mode caller who is using an 8-bytes-per-entry parameter list specifies PLIST8ARALETS=YES, the system builds a parameter list with the 8-byte addresses at the beginning of the list and their associated 4-byte ALETs following the addresses.

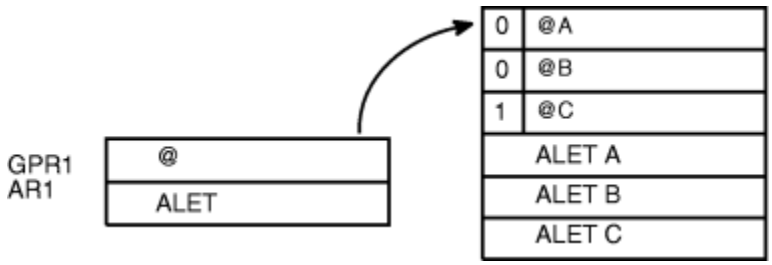


Figure 1. Sample User Parameter List for Callers in AR Mode

For information about linkage conventions, see the chapter in *z/OS MVS Programming: Assembler Services Guide*.

Telling the system about the execution environment

To generate code that is correct for the environment in which the program runs, some macros need to know one or more of the following characteristics about that environment:

- The addressing mode (AMODE) at the time the macro is issued
- The ASC mode of the program at the time the macro is issued
- The architectural level in which the program runs

For macros that are sensitive to their environment, use the SYSSTATE macro to define the environment. During the assembly stage, SYSSTATE sets one or more global symbols. Later, in your source code, the macro checks the global symbols and generates the correct code, which might mean avoiding using a z/Architecture® instruction or an access register. Table 1 lists MVS macros and identifies macros that need to know the environmental characteristics.

IBM recommends you issue the SYSSTATE macro before you issue other macros. Once a program has issued SYSSTATE, there is no need to reissue it, unless the program switches from one AMODE to another or one ASC mode to another or has code paths that are isolated according to architecture level or operating system release. If you switch AMODE or ASC mode to a different architecture code path, issue SYSSTATE immediately after the switch to indicate the new state. In general, specify SYSSTATE ARCHLVL=2, and switch to SYSSTATE ARCHLVL=3 before issuing macros in sections of code that only run when z/OS 2.1 capabilities are available. If you do not issue the SYSSTATE macro, the system assumes the macro is issued as follows:

- In AMODE other than 64-bit
- In primary ASC mode
- Usually, in ESA/390 architectural level (but may assume z/Architecture level since all supported z/OS releases require z/Architecture level)

Table 1 describes the relevant characteristics, the corresponding parameters on the SYSSTATE macro, and the global symbols the macro checks.

Table 2. Execution environment characteristics and corresponding SYSSTATE parameters and global symbols

Characteristic	Parameter on SYSSTATE	Global symbol
AMODE of 64-bit, or either 24-bit or 31-bit	AMODE64=YES or NO	&SYSAM64
Primary or AR ASC mode	ASCENV=P or AR	&SYSASCE
Architectural level of z/Architecture	ARCHLVL=0, 1, 2, 3 or OSREL	&SYSALVL
Operating system release	ZOSV*v*R*r*	&SYSOSREL

You can issue the SYSSTATE macro with the TEST parameter in your own user-written macro to allow your macros to generate code appropriate for their execution environment.

Callable services do not check the global symbols described in this topic. To determine whether a callable service is sensitive to the AMODE, ASC mode, or the Architecture level, see the description of the individual callable service.

In early releases of MVS, the SPLEVEL macro performs a function similar to SYSSTATE. The SPLEVEL macro identifies the level of the operating system, so that you can tune a macro expansion based on that level. You can use this where macro expansions change incompatibly. Because SPLEVEL applies to levels that the system no longer supports, it is not described in this topic.

Specifying a macro version number

Often there is more than one version of a macro, differentiated by additional parameters or new or expanded function. For example, version 1 of the IXGCONN macro provides a connection to a log stream, while version 2 adds new parameters in support of resource manager programs. This is different than using the SPLEVEL macro to select a macro version level to solve problems of downward compatibility.

You can request a specific version of a macro based on the parameters you need to use in your application, but you should also be attuned to the storage constraints of the program. The version of a macro might affect the length of the parameter list generated when the macro is assembled, because when you add new parameters to a macro, the parameter list must be large enough to fit them. The size of the parameter list might grow from release to release of z/OS, perhaps affecting the amount of storage your program needs.

How to request a macro version using PLISTVER

Many macros that have one or more versions supply the PLISTVER parameter. For those that do, use the PLISTVER parameter to request a version of the macro. PLISTVER is the only parameter allowed on the list form of a macro (MF), and it determines which parameter list the system generates. PLISTVER is optional. If you omit it, the system generates a parameter list for the lowest version that will accommodate the parameters specified. This is the IMPLIED_VERSION default. Note that on the list form, the default will cause the smallest parameter list to be created.

You can also code a specific version number using *plistver*, or specify MAX:

- You can use ***plistver*** to code a decimal value corresponding to the version of the macro you require. The decimal value you provide determines the amount of storage allotted for the parameter list.
- You can use **MAX** to request that the system generate a parameter list for the highest version number currently available. The amount of storage allotted for the parameter list will depend on the level of the system on which the macro is assembled.

  IBM recommends, if your program can tolerate additional growth, that you always specify PLISTVER=MAX on the list form of the macro. MAX ensures that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form when both forms are assembled using the save level of the system.

Hints for using PLISTVER

There are some general considerations that you should keep in mind when specifying the version of a macro with PLISTVER:

- If PLISTVER is omitted, the macro generates a parameter list of the lowest version that allows all the parameters specified to be processed.
- If you code PLISTVER=*n* and then specify any version '*n*+1' parameter, the macro will not assemble.
- If you code PLISTVER=*n* and do not specify any version '*n*' parameter, the macro will generate a version '*n*' parameter list.
- If you are using the standard form of the macro (MF=S), there is no reason you need to code the PLISTVER parameter.

- Not all macros have the same version numbers. The version numbers need not be contiguous.

The PLISTVER parameter appears in the syntax diagram and in the parameter descriptions. Within each macro description, the PLISTVER parameter description specifies the range of values and lists the parameters applicable for each version of the macro.

Register use

Some services require that the caller place information in specific general purpose registers (GPRs) or access registers (ARs) prior to issuing the service. If a service has such a requirement, the “Input Register Information” topic for the service provides that information. The topic lists only those registers that have a requirement. If a register is not specified as having a requirement, then the caller does not have to place any information in that register unless using it in register notation for a particular parameter, or using it as a base register.

Once the caller issues the service, the system can change the contents of one or more registers, and leave the contents of other registers unchanged. When control returns to the caller, each register contains one of the following values or has the following status:

- The register content is preserved and is the same as it was before the service was issued.
- The register contains a value placed there by the system for the caller's use. Examples of such values are return codes and tokens.
- The system used the register as a work register. Do not assume that the register content is the same as it was before the service was issued.

Unless otherwise defined by the individual interface, the calling program expects upon return:

- The low halves (Bits 32-63) of GPRs 0, 1, 14, 15 are not preserved; the low halves of GPRs 2-13 are preserved.
- The high halves (Bits 0-31) of GPRs 0, 1, and 15 are not preserved; the high halves of GPRs 2-14 are preserved.
- When GPR 0, 1, 14, and/or 15 is defined as unchanged, unless otherwise stated by the individual interface, that definition applies only to the low halves of those registers.
- ARs 0,1, 14, 15 are not preserved; ARs 2-13 are preserved.

For more information about linkage conventions for a calling program’s registers, see the "Saving the calling program’s registers" topic in the "Linkage conventions" chapter in *z/OS MVS Programming: Assembler Services Guide*.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Many macros require that the caller have a program base register and assembler USING instruction in effect when issuing the macro; that is, the caller must have *program addressability*. AR mode programs also require that the AR associated with the caller's base GPR be set to zero. **IBM recommends** the following:

- When issuing a macro, the caller should always have program addressability in effect.
- When establishing addressability, the caller should use only registers 2 through 12.

Many macros can take advantage of relative branching when they are used with the IEABRC macro or with SYSSTATE ARCHLVL=1 or SYSSTATE ARCHLVL=2, if they are running on z/OS. If relative branching is used, the caller might then need addressability only to the static data portion of the program, and not to the executable code.

Handling return codes and reason codes

Most of the services described in this information provide return codes and reason codes. Return and reason codes indicate the outcome of the service in one of the following ways:

- Successful completion: you do not need to take any action.
- Successful or partially successful completion, with additional information supplied: you should evaluate the additional information in light of your particular program and determine if you need to take any action.
- Unsuccessful completion: some type of error has occurred, and you must take some action to correct the error.

The errors that cause unsuccessful completion fall into three broad categories:

Program errors
Errors that your program causes: you can correct these.

Environmental errors
Errors not caused directly by your program; rather, your program's request caused a limit to be exceeded, such as a storage limit, or the limit on the size of a particular data set. You might or might not be able to correct these.

System errors
Errors caused by the system: your program did nothing to cause the error, and you probably cannot correct these.

In some cases, a return or reason code can result from some combination of these errors.

The return and reason code descriptions for the services in this information indicate whether the error is a program error, an environmental error, a system error, or some combination. Whenever possible, the return and reason code descriptions give you a specific action that you can take to fix the error.

IBM recommends that you read all the return and reason codes for each service that your program issues. You can then design your program to handle as many errors as possible. When designing your program, you should allow for the possibility that future releases of MVS might add new return and reason codes to a service that your program issues.

Handling program errors

The actions to take in the case of program errors are usually straightforward. Typical examples of program errors are:

1. Breaking one of the rules of the service. For example:
   - Passing parameters that are either in the wrong format or not valid
   - Violating one of the environment requirements (addressing mode, locking requirements, dispatchable unit mode, and so on)
   - Providing insufficient storage for information to be returned by the system.
2. Causing errors related to the parameter list. For example:
   - Coding an incorrect combination of parameters
   - Coding one or more parameters on the service incorrectly
   - Inadvertently overlaying an area of the parameter list storage
   - Inadvertently destroying the pointer to the parameter list.
3. Requesting a service or function for which the calling program is not authorized, or which is not available on the system on which the program is running.

In each of the first two cases, you can correct your program. For completeness, the return and reason code descriptions give you specific actions to perform, even when it might seem obvious what the action should be.

In the third case, you might have to contact your system administrator or system programmer to obtain the necessary authorization, or to request that the service or function be made available on your system, and the return or reason code description asks you to take that step.

Note: Generally, the system does not take dumps for errors that your program causes when issuing a system service. If you require such a dump, then it is your responsibility to request one in your recovery

routine. See the topic on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about writing recovery routines.

Handling environmental and system errors

With environmental errors, often your first action should be to rerun your program or retry the request one or more times. The following are examples of environmental errors where rerunning your program or retrying the request is appropriate:

- The request being made through the service exceeds some internal system limit. Sometimes, rerunning your program or retrying the request results in successful completion. If the problem persists, it might be an indication of a larger problem requiring you to consult your system programmer, or possibly IBM support personnel. Your system programmer might be able to tune the system or cancel users so that the limit is no longer exceeded.
- The request exceeds an installation-defined limit. If the problem persists, the action might be to contact your system programmer and request that a specification in an installation exit or parmlib member be modified.
- The system cannot obtain storage, or some other resource, for your request. If the problem persists, the action might be to check with the operator to see if another user in the installation is causing the problem, or to see if the entire installation is experiencing storage constraint problems.

You might be able to design your program to anticipate certain environmental errors and handle them dynamically.

With system errors, as with environmental errors, often your first action should be to rerun your program or retry the request one or more times. If the problem persists, you might have to contact IBM support personnel.

Whenever possible for environmental and system errors, the return or reason code description gives you either a specific action you can take, or a list of recommended actions you can try.

For some errors, providing a specific action is not possible, because the action you should take depends on your particular application, and on what is happening in your installation. In those cases, the return or reason code description gives you one or more possible causes of the error to help you to determine what action to take.

Some system errors result in return and reason codes that are provided for IBM diagnostic purposes only. In these cases, the return or reason code description asks you to record the information and provide it to the appropriate IBM support personnel.

Using X-macros

Some MVS services support callers in both primary and AR ASC mode. When the caller is in AR mode, macros must generate larger parameter lists; the increased size of the list reflects the addition of ALETs to qualify addresses, as described under ALET qualification. For some MVS macros, two versions of a particular macro are available: one for callers in primary mode and one for callers in AR mode. The name of the macro for the AR mode caller is the same as the name of the macro for primary mode callers, except the AR mode macro name ends with an “X”. This information refers to these macros as **X-macros**.

The authorized X-macros are:

- ATTACHX
- ESTAEX
- SDUMPX
- SYNCHX

The only way these macros know that a caller is in AR mode is by checking the global symbol that the SYSSTATE macro sets. Each of these macros (and corresponding non-X-macro) checks the symbol. If SYSSTATE ASCENV=AR has been issued, the macro issues code that is valid for callers in AR mode. If it

has not been issued, the macro generates code that is not valid for callers in AR mode. When your program returns to primary mode, use the SYSSTATE ASCENV=P macro to reset the global symbol.

IBM recommends that you use the X-macro regardless of whether your program is running in primary or AR mode. However, you should consider the following before deciding which macro to use:

The rules for using all X-macros, except ESTAEX, are:

- Callers in primary mode can invoke either macro.

  Some parameters on the X-macros, however, are not valid for callers in primary mode. Some parameters on the non-X-macros are not valid for callers in AR mode. Check the macro descriptions for these exceptions.
- Callers in AR mode should issue the X-macros.

  If a caller in AR mode issues the non-X-macro, the system substitutes the X-macro and sends a message describing the substitution.

IBM recommends you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, or your program requires a branch entry. In these cases, you should use ESTAE.

Macro forms

You can code most macros in three forms: standard, list, and execute. Some macros also have a modify form. When you code a macro, you use the MF parameter to select one of the forms. The list, execute and modify forms are for reenterable programs that need to change values in the parameter list of the macro. The standard form is for programs that are not reenterable, or for programs that do not change values in the parameter list.

When a program wants to change values in the parameter list of a macro, it can make the change dynamically.

However, using the standard form and changing the parameter list dynamically might cause errors. For example, after storing a new value into the inline, standard form of the parameter list, a reenterable program operating under a given task might be interrupted by the system before the program can invoke the macro. In a multiprogramming environment, another task can use the same reenterable program, and that task might change the inline parameter list again before the first task regains control. When the first task regains control, it invokes the macro. However, the inline parameter list now has the wrong values.

Through the use of the different macro forms, a program that runs in a multiprogramming environment can avoid errors related to reenterable programs. The techniques required for using the macro forms, however, are different for some macros, called alternative list form macros, than for most other macros. For the alternative list form macros, the list form description notes that different techniques are required and refers you to the information under Alternative list form macros.

Conventional list form macros

With conventional list form macros, you can use the macro forms as follows:

1. Use the list form of the macro, which expands to the parameter list. Place the list form in the section of your program where you keep non-executable data, such as program constants. Do not code it in the instruction stream of your program.
2. In the instruction stream, code a GETMAIN or a STORAGE macro to obtain some virtual storage.
3. Code a move character instruction that moves the parameter list from its non-executable position in your program into the virtual storage area that you obtained.
4. For macros that have a modify form, you can code the modify form of the macro to change the parameter list. Use the address parameter of the modify form to reference the parameter list in the virtual storage area that you obtained. Thus, the parameter list that you change is the one in the virtual storage area obtained by the GETMAIN or STORAGE macro.
5. Invoke the macro by issuing the execute form of the macro. Use the address parameter of the execute form to reference the parameter list in the virtual storage area that you obtained.

With this technique, the parameter list is safe even if the first task is interrupted and a second task intervenes. When the program runs under the second task, it cannot access the parameter list in the virtual storage of the first task.

Alternative list form macros

Certain macros, called alternative list form macros, require a somewhat different technique for using the list form. With these macros, you do not move the area defined by the list form into virtual storage that you have obtained; instead, you place the area defined by the list form into a DSECT. Also, it is the list form, not the execute form, that you use to specify the address parameter that identifies the address of the storage for the parameter list. Note that no modify form is available for these macros.

You can use the macro forms for the alternative list form macros as follows:

1. Use the list form of the macro to define an area of storage that the execute form can use to store the parameters. As with other macros, do not code the list form in the instruction stream of your program.
2. In the instruction stream, code a GETMAIN or a STORAGE macro to obtain virtual storage for the list form expansion.
3. Place the area defined by the list form into a DSECT that maps a portion of the virtual storage you obtained.
4. Invoke the macro by issuing the execute form of the macro. The address parameter specified on the list form references the parameter list in the virtual storage area that you obtained.

Coding the macros

In this information, each macro description includes a syntax diagram near the beginning of the macro description. The diagram shows how to code the macro. The syntax diagram does not explain the meanings of the parameters; the meanings are explained in the parameter descriptions that follow the syntax diagram. For most macros, the syntax diagrams are in a tabular format; however, some newer macros might have syntax diagrams in the railroad track format.

The syntax tables assume that the standard begin, end, and continue columns are used. Thus, column 1 is assumed as the begin column. To change the begin, end, and continue columns, use the ICTL instruction to establish the coding format you want to use. If you do not use ICTL, the assembler recognizes the standard columns. For more information about coding the ICTL instruction, see High Level Assembler and Toolkit Feature in IBM Documentation (www.ibm.com/docs/en/hla-and-tf/1.6).

Figure 1 shows a sample macro called TEST and summarizes all the coding information that is available for it. The table is divided into three zones, A, B, and C.

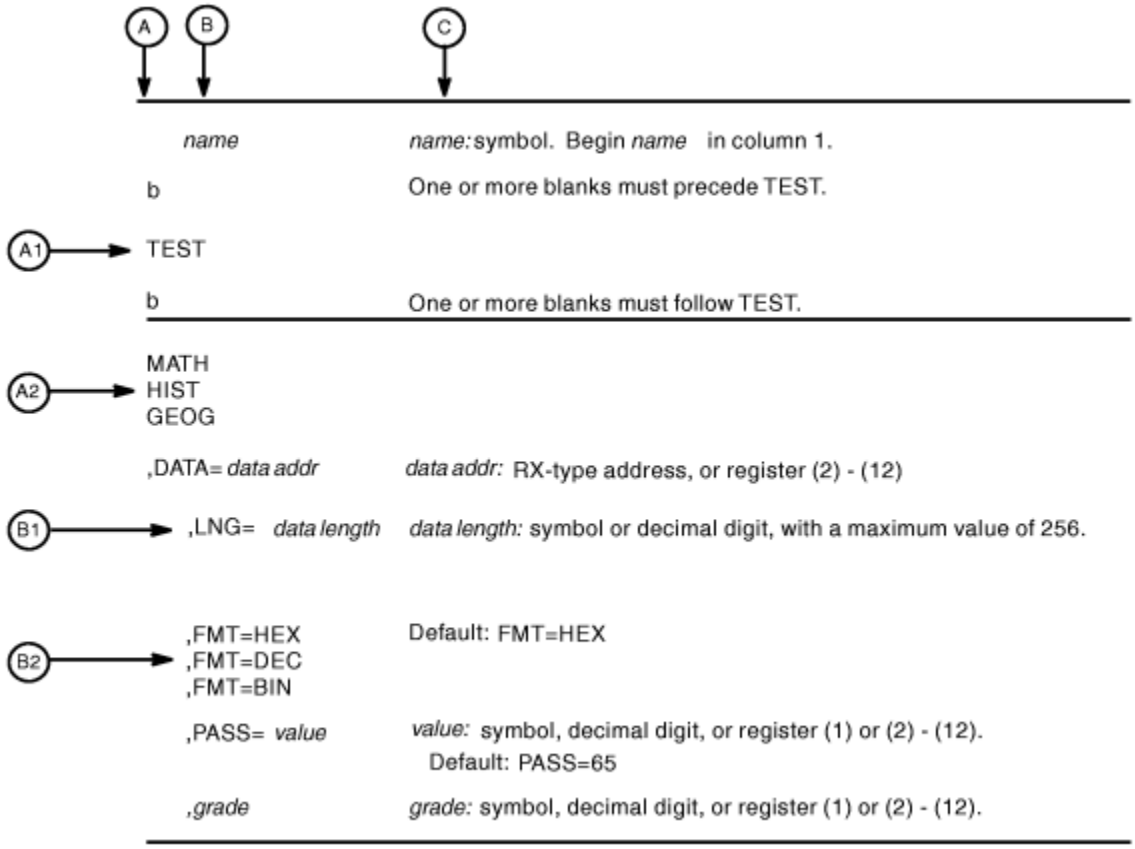


Figure 2. Sample tabular syntax diagram for the TEST macro

- Column one of the table contains zones A and B. Zone A begins at the left margin; zone B is indented from the left margin by one or more blank spaces. Column two of the table contains zone C.
- Zone A and zone B contain those parameters that are allowed for the macro. Zone A contains those parameters that are required; zone B contains those parameters that are optional.
- If a parameter appears on a single line in the diagram (that is, a line whose preceding line and following line are both blank), as shown in A1 and B1, then that is the only available choice for the particular parameter.
- If two or more parameters appear on adjacent lines (that is, with no intervening blank lines), as shown in A2 and B2, the parameters on those lines are mutually exclusive, that is, you can code any one of those parameters.
- A further distinction is made between mandatory and optional parameters. The parameter descriptions that follow the syntax table clearly identify those parameters which are optional.
- Zone C (which is the second column in the syntax table), provides additional information about coding the macro.

When substitution of a variable is indicated in zone C, the following classifications are used:

Variable
Classification

symbol
Any symbol valid in the assembler language. The symbol can be as long as the supported maximum length of a name entry in the assembler you are using.

Decimal digit
Any decimal digit up to and including the value indicated in the parameter description. If both symbol and decimal digit are indicated, an absolute expression is also allowed.

Register (2) - (12)
One of general purpose registers 2 through 12, specified within parentheses, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. You can designate the register symbolically or with an absolute expression.

Register (0)
General purpose register 0, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (0) only.

Register (1)
General purpose register 1, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (1) only.

Register (15)
General purpose register 15, previously loaded with the right-adjusted value or address indicated in the parameter description. You must set the unused high-order bits to zero. Designate the register as (15) only.

RX-type address
Any address that is valid in an RX-type instruction (for example, LA).

RS-type address
Any address that is valid in an RS-type instruction (for example, STM).

RS-type name
Any name that is valid in an RS-type instruction (for example, STM).

A-type address
Any address that can be written in an A-type address constant.

Default
A value that is used in default of a specified value; that is, the value the system assumes if the parameter is not coded.

Rules for parameters: Use the parameters to specify the services and options to be performed, and write them according to the following rules:

- If the selected parameter is written in all capital letters (for example, MATH, HIST, or FMT=HEX), code the parameter exactly as shown.
- If the selected parameter is written in italics (for example, *grade*), substitute the indicated value, address, or name.
- If the selected parameter is a combination of capital letters and italics separated by an equal sign (for example, DATA=*data addr*), code the capital letters and equal sign as shown, and then make the indicated substitution for the italicized portion.
- Read the table from top to bottom.
- Code commas and parentheses exactly as shown.
- Positional parameters (parameters without equal signs) appear first; you must code them in the order shown. You may code keyword parameters (parameters with equal signs) in any order.
- If you select a parameter, read the second column (zone C) before proceeding to the next parameter. The second column often contains coding restrictions for the parameter.

Continuation lines

You can continue the parameter field of a macro on one or more additional lines according to the following rules:

- Enter a continuation character (not blank, and not part of the parameter coding) in column 72 of the line.
- Continue the parameter field on the next line, starting in column 16. All columns to the left of column 16 must be blank.

You can code the parameter field being continued in one of two ways. Code the parameter field through column 71, with no blanks, and continue in column 16 of the next line; or truncate the parameter field by a comma, where a comma normally falls, with at least one blank before column 71, and then continue in column 16 of the next line. Figure 1 shows an example of each method.

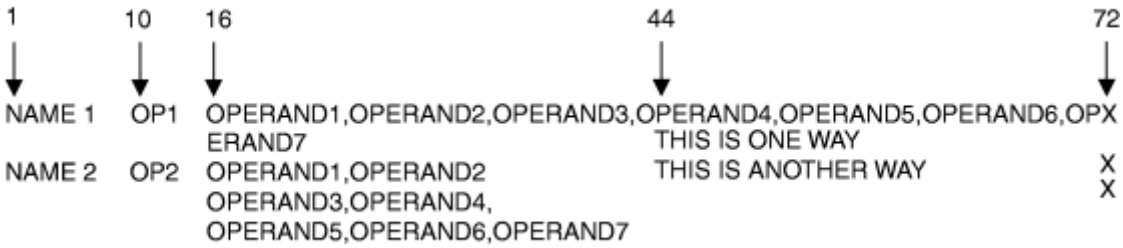


Figure 3. Continuation Coding

Coding the callable services

A callable service is a programming interface that uses the CALL macro to access system services. To code a callable service, code the CALL macro followed by the name of the callable service, and a parameter list; for example:

```
CALL service,(parameter list)
```

The syntax diagram for the sample callable service SCORE:

Syntax	Description
CALL SCORE	,(test_type ,level ,data ,format_option ,return_code)

Considerations for coding callable services are:

- You must code all the parameters in the parameter list because parameters are positional in a callable service interface. That is, the function of each parameter is determined by its position with respect to the other parameters in the list. Omitting a parameter, therefore, assigns the omitted parameter's function to the next parameter in the list.
- You must place values explicitly into all input parameters, because callable services do not set default values.
- You can use the list and execute forms of the CALL macro to preserve your program's reentrancy.

Including equate (EQU) statements

IBM supplies sets of equate (EQU) statements for use with some callable services. These statements, which you may optionally include in your source code, provide constants for use in your program. IBM provides the statements as a programming convenience to save you the trouble of coding the definitions yourself.

Note: Check the “Programming Requirements” section of the individual service description to determine if the equate statements are available for the callable service you are using. If the equate statements are

available, that section will also provide a list of the statements that are provided, along with a description of how to include them in your program.

Link-editing linkage-assist routines

Linkage-assist routines provide the connection between your program and the system services that your program requests. When using callable services, link-edit the appropriate linkage-assist routines into your program module so that, during execution, the linkage-assist routines can resolve the address of, and pass control to, the requested system services. You can also dynamically link to linkage-assist routines as an alternative to link-editing. For example, issue the LOAD macro for the linkage-assist routine, then issue a CALL to the loaded addresses.

To invoke the linkage-editor or binder, code JCL as in the following example:

```
//userid   JOB 'accounting-info','name',CLASS=x,
//  MSGCLASS=x,NOTIFY=userid,MSGLEVEL=(1,1),REGION=4096K
//LINKSTEP EXEC PGM=HEWL,
//  PARM='LIST,LET,XREF,REFR,RENT'
//SYSPRINT DD   SYSOUT=x
//SYSLMOD  DD   DSN=userid.LOADLIB,DISP=OLD
//SYSLIB   DD   DSN=SYS1.CSSLIB,DISP=SHR
//OBJLIB   DD   DSN=userid.OBJLIB,DISP=SHR
//SYSUT1   DD   UNIT=SYSDA,SPACE=(TRK,(5,2))
//SYSLIN   DD   *
  INCLUDE OBJLIB(userpgm)
  ENTRY userpgm
  NAME userpgm(R)
/*
```

Note: Omitting NCAL from the linkedit parameters (as the example shows) and specifying SYS1.CSSLIB in the //SYSLIB statement, as shown, causes the addresses of all required linkage-assist routines to be automatically resolved. This statement saves you the trouble of having to specify individual linkage-assist routines in INCLUDE statements.

Service summary

Table 1 lists services described in the following:

- *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*
- *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*
- *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*
- *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*.

For each service, the table indicates:

- Whether a program in AR ASC mode can issue the service
- Whether a program in cross memory mode can issue the service
- Whether the macro checks the SYSSTATE global macro variables
- Whether the macro can be issued in 64-bit addressing mode

Note:

1. A program running in primary ASC mode when PASN=HASN=SASN can issue any of the services listed in the table.
2. Cross memory mode means that at least one of the following conditions is true:

   PASN¬=SASN
   The primary address space (PASN) and the secondary address space (SASN) are different.

   PASN¬=HASN
   The primary address space (PASN) and the home address space (HASN) are different.

   SASN¬=HASN
   The secondary address space (SASN) and the home address space (HASN) are different.

For more information about functions that are available to programs in cross memory mode, see *z/OS MVS Programming: Extended Addressability Guide*.

3. Callable services do not check the SYSSTATE or SPLEVEL global variables.

Table 3. Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
ALESERV	Yes	Yes	No	No
ASCRE	Yes	Yes	Yes	No
ASDES	Yes	Yes	Yes	No
ASEXT	Yes	Yes	No	No
ATSET	No	Yes	Yes	No
ATTACH	Yes (See note 1)	No	Yes	No
ATTACHX	Yes	No	Yes	Yes
AXEXT	No	Yes	Yes	No
AXFRE	No	Yes	Yes	No
AXRES	No	Yes	Yes	No
AXREXX	No	Yes	Yes	Yes
AXSET	No	Yes	Yes	No
BPXEKDA	Yes	No	Yes	No
BPXESMF	Yes	No	Yes	No
CALLDISP	No	Yes	No	Yes
CALLRTM	No	Yes (See note 2)	No	No
CHANGKEY	No	Yes	No	No
CIRB	No	No	No	No
CMDAUTH	No	No	No	No
CNZMXURF	No	Yes	No	No
CNZTRKR	No	Yes	No	No
COFCREAT	Yes	Yes	Yes	No
COFDEFIN	Yes	Yes	Yes	No
COFIDENT	Yes	Yes	Yes	No
COFNOTIF	Yes	Yes	Yes	No
COFPURGE	Yes	Yes	Yes	No
COFREMOV	Yes	Yes	Yes	No
COFRETRI	Yes	Yes	Yes	No
COFSDONO	No	No	Yes	No
CONFCHG	No	No	Yes	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
CPF	No	No	No	No
CPOOL	No	Yes	Yes	No
CPUTIMER	No	Yes	Yes	No
CSRSI	No	Yes	No	No
CSRUNIC	Yes	Yes	No	No
CSVAPF	Yes (See note 11)	Yes (See note 12)	Yes	No
CSVDYNEX	Yes (See note 13)	Yes (See note 14)	Yes	No
CTRACE	No	No	Yes	No
CTRACECS	Yes	No	Yes	No
CTRACEWR	Yes	Yes	Yes	No
DATOFF	Yes	No	No	No
DEQ	No	Yes	Yes	Yes
DIV	Yes	No	Yes	No
DOM	No	No	No	Yes
DSPSERV	Yes	Yes	Yes	Yes
DYNALLOC	No	No	No	Yes
EDTINFO	No	Yes	Yes	Yes
ENFREQ	No	No	No	No
ENQ	No	Yes	Yes	Yes
ESPIE	No	No	No	Yes
ESTAE (See note 3)	No	No	Yes	No
ESTAEX	Yes	Yes	Yes	Yes
ETCON	No	Yes	Yes	No
ETCRE	No	Yes	Yes	No
ETDEF	Yes	Yes	No	No
ETDES	No	Yes	Yes	No
ETDIS	No	Yes	Yes	No
EVENTS	No	No	No	No
EXTRACT	No	No	No	No
FESTAE	No	No	No	No
FREEMAIN	Yes (See note 4)	Yes	Yes	Yes
GETDSAB	No	No	Yes	No
GETMAIN	Yes (See note 4)	Yes	Yes	Yes

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
GQSCAN	No	Yes	No	No
GTRACE	No	Yes	No	Yes
HSPSERV	Yes	Yes (See note 5)	(See note 6)	No
IARCP64	Yes	Yes	Yes	Yes
IARR2V	Yes	Yes	No	No
IARSUBSP	Yes	Yes	Yes	No
IARST64	Yes	Yes	Yes	Yes
IARVSERV	Yes	Yes	Yes	No
IARV64	Yes	Yes	Yes	Yes
IAZXCTKN	Yes	Yes	Yes	No
IAZXJSAB	Yes	Yes (See note 15)	Yes	No
IEAARR	Yes	Yes	Yes	Yes
IEAFP	Yes	Yes	Yes	No
IEALSQRY	Yes	Yes	Yes	No
IEAMETR	Yes	Yes	Yes	No
IEAMRMF3	No	Yes	No	No
IEAMSCHD	Yes	Yes	Yes	No
IEANTCR	Yes	Yes	N/A	No
IEANTDL	Yes	Yes	N/A	No
IEANTRT	Yes	Yes	N/A	No
IEARBUP	Yes	Yes	Yes	No
IEATDUMP	Yes	No	Yes	No
IEATEDS	Yes	Yes	Yes	No
IEATXDC	Yes	Yes	Yes	Yes
IEAVAPE	No	Yes	No	No
IEAVAPE2	No	Yes	No	No
IEAVDPE	No	Yes	No	No
IEAVDPE2	No	Yes	No	No
IEAVPSE	No	Yes	No	No
IEAVPSE2	No	Yes	No	No
IEAVRLS	No	Yes	No	No
IEAVRLS2	No	Yes	No	No
IEAVRPI	No	Yes	No	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
IEAVRPI2	No	Yes	No	No
IEAVTPE	No	Yes	No	No
IEAVXFR	No	Yes	No	No
IEAVXFR2	No	Yes	No	No
IEA4APE	No	Yes	No	Yes
IEA4APE2	No	Yes	No	Yes
IEA4DPE	No	Yes	No	Yes
IEA4DPE2	No	Yes	No	Yes
IEA4PSE	No	Yes	No	Yes
IEA4PSE2	No	Yes	No	Yes
IEA4RLS	No	Yes	No	Yes
IEA4RLS2	No	Yes	No	Yes
IEA4RPI	No	Yes	No	Yes
IEA4RPI2	No	Yes	No	Yes
IEA4TPE	No	Yes	No	Yes
IEA4XFR	No	Yes	No	Yes
IEA4XFR2	No	Yes	No	Yes
IEECMDS	Yes	Yes	Yes	No
IEEQEMCS	Yes	Yes	Yes	No
IEEVARYD	No	No	Yes	No
IEFPPSCN	No	No	Yes	No
IEFQMREQ	No	No	No	No
IEFSSI	Yes	No	No	No
IEFSSVT	Yes	No	No	No
IEFSSVTI	Yes	Yes	No	No
IFAQUERY	Yes	Yes	No	No
IOCINFO	Yes	Yes	No	No
IOSADMF	No	No	Yes	No
IOSCAPF	No	Yes (See note 7)	Yes	No
IOSCAPU	Yes	Yes (See note 7)	Yes	No
IOSCDR	No	No	Yes	No
IOSCHPD	Yes	Yes	Yes	No
IOSCMXA	No	Yes (See note 7)	Yes	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
IOSCMXR	No	Yes (See note 7)	Yes	No
IOSDCXR	No	Yes (See note 7)	Yes	No
IOSENQ	Yes	Yes	Yes	No
IOSINFO	No	No	No	No
IOSLOOK	No	No	No	No
IOSPTHV	No	No	Yes	No
IOSSPOF	No	Yes	Yes	Yes
IOSUPFA	No	Yes	Yes	No
IOSUPFR	No	Yes	Yes	No
IOSVRYSW	Yes	Yes	Yes	No
IOSWITCH	Yes	Yes	Yes	No
IOSZHPF	Yes	Yes	Yes	No
IRDFSD	Yes	Yes	Yes	No
IRDFSDU	Yes	Yes	Yes	No
ISGADMIN	Yes	Yes	Yes	Yes
ISGECA	Yes	Yes	Yes	Yes
ISGENQ	Yes	Yes	Yes	Yes
ISGLCRT (See note 16)	No	Yes	N/A	No
ISGLID (See note 16)	No	Yes	N/A	Yes
ISGLOBT	No	Yes	N/A	No
ISGLREL	No	Yes	N/A	No
ISGLPRG	No	Yes	N/A	No
ISGQUERY	Yes	Yes	Yes	Yes
ITTFMTB	No	No	No	No
ITZXFILT	No	Yes	Yes	No
IWMCLSFY	No	Yes	Yes	No
IWMCONN	No	Yes	Yes	No
IWMDISC	No	Yes	Yes	No
IWMECQRY	No	Yes	Yes	No
IWMECREA	No	Yes	Yes	No
IWMEDELE	No	Yes	Yes	No
IWMMABNL	No	Yes	No	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
IWMMCHST	No	Yes	No	No
IWMMCREA	No	Yes	Yes	No
IWMMDELE	No	Yes	Yes	No
IWMMEXTR	No	Yes	Yes	No
IWMMINIT	No	Yes	No	No
IWMMNTFY	No	Yes	Yes	No
IWMMRELA	No	Yes	Yes	No
IWMMSWCH	No	Yes	Yes	No
IWMMXFER	No	Yes	No	No
IWMPQRY	Yes	Yes	Yes	No
IWMRCOLL	Yes	Yes	Yes	No
IWMRPT	No	Yes	Yes	No
IWMRQRY	Yes	Yes	Yes	No
IWMSRDRS	No	Yes	Yes	No
IWMSRSRG	No	Yes	Yes	No
IWMSRSRS	No	Yes	Yes	No
IWMWMCON	No	Yes	Yes	No
IWMWQRY	Yes	Yes	Yes	No
IWMWQWRK	No	Yes	Yes	No
IXCCREAT	Yes	Yes	Yes	No
IXCDELET	Yes	Yes	Yes	No
IXCJOIN	Yes	No	Yes	No
IXCLEAVE	Yes	No	Yes	No
IXCMG	Yes	Yes	Yes	No
IXCMOD	Yes	Yes	Yes	No
IXCMSGI	Yes	No	Yes	No
IXCMSGO	Yes	Yes	Yes	No
IXCQUERY	Yes	Yes	Yes	No
IXCQUIES	Yes	No	Yes	No
IXCSETUS	Yes	Yes	Yes	No
IXCTERM	Yes	Yes	Yes	No
IXGBRWSE	Yes	Yes	Yes	Yes
IXGCONN	Yes	Yes	Yes	Yes

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
IXGDELET	Yes	Yes	Yes	Yes
IXGWRITE	Yes	Yes	Yes	Yes
LLACOPY	No	No	Yes	No
LOAD	Yes	No	No	Yes
LOADWAIT	No	Yes	Yes	No
LOCASCB	Yes	Yes	Yes	No
LXFRE	No	Yes	Yes	No
LXRES	No	Yes	Yes	No
MCSOPER	Yes	No	Yes	No
MCSOPMSG	Yes	No	Yes	No
MGCR	No	No	No	No
MGCRE	No	No	No	No
MIHQUERY	Yes	No	Yes	No
MODESET	No	Yes	No	Yes
NIL	Yes	Yes	Yes	No
NMLDEF	No	No	No	No
NUCLKUP	No	No	No	No
OIL	Yes	Yes	Yes	No
OUTADD	No	No	No	No
OUTDEL	No	No	No	No
PCLINK	No	Yes	No	No
PGANY	No	No	No	No
PGFIX	No	Yes	No	No
PGFIXA	No	No	No	No
PGFREE	No	Yes	No	No
PGFREEA	No	No	No	No
PGSER	Yes (See note 8)	Yes (See note 8)	No	Yes
POST	No	Yes	No	Yes
PTRACE	No	Yes	No	No
PURGEDQ	No	No	No	No
QEDIT	No	No	No	No
RESERVE	No	No	No	Yes
RESMGR	Yes	Yes	No	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
RESUME	No	Yes	No	No
RISGNL	No	Yes	No	No
SCHEDIRB	Yes	No	Yes	No
SCHEDULE	Yes	Yes	Yes	No
SCHEDXIT	No	Yes	No	No
SDUMP	Yes (See note 1)	Yes (See note 9)	Yes	No
SDUMPX	Yes	Yes (See note 9)	Yes	Yes
SETFRR	Yes	Yes	Yes	No
SETLOCK	Yes	Yes	Yes	No
SETRP	Yes	Yes	Yes	Yes
SJFREQ	No	Yes	No	No
SPIE	No	No	No	No
SPOST	No	No	No	No
SRBSTAT	No	Yes	No	No
SRBTIMER	No	No	No	No
STATUS	Yes	Yes	No	No
STORAGE	Yes	Yes	No	Yes
SUSPEND	No	Yes	No	No
SVCUPDTE	No	No	No	No
SWAREQ	No	No	No	No
SWBTUREQ	No	No	No	No
SYMREC	No	Yes	Yes	No
SYNCH	Yes (See note 1)	No	Yes	No
SYNCHX	Yes	No	Yes	Yes
SYSEVENT	No	No	No	No
TCBTOKEN	Yes	Yes	No	No
TCTL	No	No	No	No
TESTAUTH	No	No	No	No
TIMEUSED	Yes (See note 10)	Yes	No	Yes
T6EXIT	No	No	No	No
UCBINFO	Yes	Yes	Yes	No
UCBLOOK	Yes	Yes	Yes	No
UCBPIN	Yes	Yes	Yes	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
UCBSCAN	Yes	Yes	Yes	No
VSMLIST	No	Yes	Yes	No
VSMLOC	No	Yes	Yes	No
VSMREGN	No	Yes	No	No
WAIT	No	Yes	No	Yes
WTL	No	No	No	No
WTO	No	No	No	Yes
WTOR	No	No	No	Yes

Notes:

1. Primary mode callers can use either macro in the following macro pairs:
   - ATTACH or ATTACHX
   - SDUMP or SDUMPX
   - SYNCH or SYNCHX

   IBM recommends that programs in AR ASC mode use the X-macros (ATTACHX, SDUMPX, and SYNCHX). If, however, a program in AR mode issues ATTACH, SDUMP, or SYNCH after issuing SYSSTATE ASCENV=AR, the system substitutes the corresponding X-macro and issues a message telling you that it made the substitution.
2. CALLRTM TYPE=MEMTERM can be issued in cross memory mode. For CALLRTM TYPE=ABTERM, see the CALLRTM macro description.
3. The only programs that can use ESTAE are programs that are in primary mode with (PASN=HASN=SASN).

   IBM recommends you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, or your program requires a branch entry. In these cases, you should use ESTAE.
4. IBM recommends that AR mode callers use the STORAGE macro instead of using GETMAIN or FREEMAIN.
5. For HSPSERV SREAD and HSPSERV SWRITE, PASN=HASN=SASN for a non-shared standard hiperspace for which an ALET is not used (that is, the HSPALET parameter is omitted).
6. If you use the HSPALET parameter, the HSPSERV macro checks SYSSTATE.
7. If the input UCB is captured, the IOSCAPF, IOSCMXA, IOSCMXR, and IOSDCXR macros can be issued in cross memory mode only if the UCB is captured in the primary address space. IOSCAPU CAPTOACT without the ASID parameter also can be issued in cross memory mode if the UCB was captured in the primary address space. IOSCAPU CAPTUCB and IOSCAPU UCAPTUCB cannot be issued in cross memory mode.
8. PGSER can be issued in AR ASC mode only if you specify BRANCH=Y. PGSER can be issued in cross memory mode only if you specify BRANCH=Y or BRANCH=SPECIAL.
9. Both SDUMP and SDUMPX can be issued in cross memory mode only if you specify BRANCH=YES.
10. Only TIMEUSED LINKAGE=SYSTEM can be issued in AR ASC mode. TIMEUSED LINKAGE=BRANCH cannot be issued in AR ASC mode.
11. For a QUERY request, CSVAPF can be issued only in primary mode. For all other requests, CSVAPF can be issued in primary or AR mode.

12. For CSVAPF with the ADD, DELETE, and DYNFORMAT requests, PASN = HASN = SASN. For CSVAPF with the QUERY, QUERYFORMAT, and LIST requests, any PASN, any HASN, any SASN.
13. For a QUERY or a CALL request with FASTPATH=YES, CSVDYNEX can be issued only in primary mode. For all other requests, CSVDYNEX can be issued in primary or AR mode.
14. For CSVDYNEX CALL, RECOVER, and QUERY requests, any PASN, any HASN, any SASN. For all other requests, PASN=HASN=SASN.
15. When the caller of the IAZXJSAB macro specifies the ASCB parameter, any PASN, any HASN, any SASN; otherwise, PASN=HASN is required.
16. The 64 bit entry names are as follows:
    - ISGLCR64
    - ISGLID64
    - ISGLOB64
    - ISGLRE64
    - ISGLPB64
    - ISGLPR64

Chapter 2. EDTINFO — Obtain eligible device table information

Description

The EDTINFO macro enables you to obtain information from the eligible device table (EDT) and to check your device specification against the information in the EDT. See *z/OS HCD Planning* and *z/OS MVS Programming: Assembler Services Guide* for further information about the EDT.

For callers only in supervisor state AND PSW key 0, the EDTINFO macro performs the following function:

- Return EDT Latch Tables (RTNEDTLT)

Note: If the RTNEDTLT function is specified, no other EDTINFO functions can be requested on the same invocation nor can the IOCTOKEN and EDTADDR keywords be specified.

For a list of functions performed by the EDTINFO macro for both unauthorized and authorized callers, see *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and any PSW key 0.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31- bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space.

Programming requirements

Callers requesting the RTNEDTLT function of the EDTINFO macro must be in 31-bit AMODE to reference the areas returned through the ELTPRI and ELTSEC pointers.

Callers requesting the RTNEDTLT function of the EDTINFO macro are required to free the storage returned through the ELTPRI and ELTSEC pointers.

Restrictions

Callers must be supervisor state and PSW key 0 in order to invoke the RTNEDTLT function.

Input register information

Before issuing the EDTINFO macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code if GPR 15 contains a return code of 04 or 08; otherwise, used as a work register by the system

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the EDTINFO macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EDTINFO.
EDTINFO	
ƀ	One or more blanks must follow EDTINFO.

Syntax	Description
RTNEDTLT	**Note:** If this function is specified, no other functions can be requested.
,ELTPRI=*eltpri*	*eltpri*: RS-type address or register (2) - (12).
,ELTSEC=*eltsec*	*eltsec*: RS-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

RTNEDTLT
Specifies that the EDTINFO service should return both the primary and secondary EDT Latch Tables.

,ELTPRI=*eltpri*
Specifies the fullword output field that will contain the address of the primary EDT Latch Table. The area returned can be mapped by IEFDELT.

ELTSEC=*eltsec*
Specifies the fullword output field that will contain the address of the secondary EDT Latch Table. The area returned can be mapped by IEFDELT.

,RETCODE=*retcode*
Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0.

Return and reason codes

When control returns from EDTINFO, GPR 15 (and *retcode addr*, if you coded RETCODE) contains one of the following hexadecimal return codes:

Return Code	Meaning
00	The requested function or functions were performed and no reason code information has been returned.
04	The requested function or functions were performed and information has been returned, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in *rsncode*, if you coded RSNCODE). **Reason Code** **Meaning** **04** Either the primary EDT Latch Table or the secondary EDT Latch Table or both EDT Latch Tables contain no entries.

Return Code	Meaning
08	There is data in the input parameter list that is not valid, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in *rsncode*, if you coded RSNCODE). **Reason Code** **Meaning** **01** The input unit name could not be found in the EDT. **02** The input device type could not be found in the EDT. **03** One or more of the input device numbers is invalid. **04** The caller did not provide sufficient storage for the returned information. **05** The MAXELIG function requires a generic device type as input, but the input specified does not represent a generic device type. **06** The caller did not request any functions. **07** The caller requested functions that are not valid **08** For a required input, the caller specified a value that is not valid. For example, other functions were specified with a function that requires no other function requests. **09** The caller was not in supervisor state and PSW key 0 for a function that requires this environment.
10	Storage could not be obtained for the request.
18	An unexpected system error occurred.

Example

Obtain the EDT Latch Tables for both the primary and secondary EDTs.

```
EDTINFO RTNEDTLT,ELTPRI=PRI_ELT_PTR,ELTSEC=SEC_ELT_PTR
```

EDTINFO - List form

Use the list form of the EDTINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses for storing the parameters.

Syntax

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros.

The list form of the EDTINFO macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EDTINFO.
EDTINFO	
ƀ	One or more blanks must follow EDTINFO.
MF=(L,*list addr*)	*list addr*: Symbol.
MF=(L,*list addr*,*attr*)	*attr*: 1- to 60-character input string
MF=(L,*list addr*,0D)	**Default**: 0D

Parameters

The parameters are explained as follows:

MF=(L,*list addr*)
MF=(L,*list addr*,*attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the EDTINFO macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

EDTINFO - Execute form

Use the execute form of the EDTINFO macro together with the list form for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the EDTINFO macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EDTINFO.

Syntax	Description
EDTINFO	
ƀ	One or more blanks must follow EDTINFO.
RTNEDTLT	**Note:** If this function is specified, no other functions can be requested.
,ELTPRI=*eltpri*	*eltpri*: RS-type address or register (2) - (12).
,ELTSEC=*eltsec*	*eltsec*: RS-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters

The parameters are explained under the standard form of the EDTINFO macro with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
Specifies the execute form of the EDTINFO macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified. NOCHECK specifies that the system does not check for required parameters and does not supply defaults for optional parameters that were not specified.

Note: When using the NOCHECK option, make sure that it is preceded by an execute or modify form invocation that specifies or defaults to the COMPLETE option. Otherwise, the parameter list might not be completely initialized.

EDTINFO - Modify form

Use the modify form of the EDTINFO macro to change parameters in the control parameter list that the system created through the list form of the macro.

Syntax

The modify form of the EDTINFO macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EDTINFO.
EDTINFO	
ƀ	One or more blanks must follow EDTINFO.
RTNEDTLT	**Note:** If this function is specified, no other functions can be requested.
,ELTPRI=*eltpri*	*eltpri*: RS-type address or register (2) - (12).
,ELTSEC=*eltsec*	*eltsec*: RS-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RX-type address or register (2) - (12).
,MF=(M,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,MF=(M,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained under the standard form of the EDTINFO macro with the following exceptions:

,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

Specifies the modify form of the EDTINFO macro.

The *list addr* parameter specifies the address of the storage area for the parameter list.

COMPLETE specifies that the system is to check for required parameters and supply defaults for optional parameters that were not specified. NOCHECK specifies that the system does not check for required parameters and does not supply defaults for optional parameters that were not specified.

Note: When using the NOCHECK option, make sure that it is preceded by an execute or modify form invocation that specifies or defaults to the COMPLETE option. Otherwise, the parameter list might not be completely initialized.

Chapter 3. ENFREQ — Listen for system events

Description

The ENFREQ macro enables an authorized program to:

- Register to be notified when an ENF-defined event occurs (ACTION=LISTEN), or
- Delete registration for notification of an ENF-defined event (ACTION=DELETE).

To listen for an event, a program issues ENFREQ with the ACTION=LISTEN parameter. When the event that the program is listening for occurs, control passes to the listener user exit routine specified on the EXIT or SRBEXIT parameter. For a list of the events for which a program can listen, see Table 1.

To stop listening for an event, a program issues ENFREQ with the ACTION=DELETE parameter to delete the listen request. When a program issues ENFREQ with the ACTION=DELETE parameter, ENF either deletes the listen request immediately if the listener user exit has completed, or waits until the listener user exit completes. Because the listener user exit might not have completed processing at the time the delete request is issued, you must not release the listener user exit's storage or any resources that may be required by the exit. ENF does not delete the user exit when it deletes a listen request. See "DELETE option" on page 62 for the syntax of a delete request.

For guidance information about how to use the ENFREQ macro and code the listener user exit routine, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

The caller of ENFREQ must do the following:

- Include the CVT, IEFENFCT, and IEFENFPM mapping macros. Specify the DSECT=YES option with the CVT mapping macro.
- Declare a fullword and label it ENFPTR.

Restrictions

None.

Input register information

Before issuing the ENFREQ macro, the caller must ensure that the following GPRs contain the specified information:

Register
Contents

13
Address of a standard 18-word save area.

Output register information

When control returns to the caller of the ENFREQ macro, the general purpose registers (GPRs) contain:

Register
Contents

0
Unchanged

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller of the ENFREQ macro, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Reason code

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

LISTEN option

Syntax

The standard form of the ENFREQ macro for ACTION=LISTEN is written as follows:

Syntax	Description

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENFREQ.
ENFREQ	
ƀ	One or more blanks must follow ENFREQ.
ACTION=LISTEN	
,CODE=*event code*	*event code*: Decimal digit or symbol.
,DTOKEN=*dtoken*	*dtoken*: RX-type address or address in register (2) - (12).
,DISABLE=NO	**Default:** DISABLE=YES
,DISABLE=YES	
,ESTBNME=*estab name*	*estab name*: RX-type address or address in register (2) - (12).
	Default:
,EXITNME=*exitrtn name*	*exitrtn name:* RX-type address or address in register (2) - (12).
	Default:
,MASEXIT=No	**Default:** MASEXIT=NO
,MASEXIT=YES	
,FLTRBLK=*filter block addr*	*filter block addr:* RX-type address or address in register (2) – (12).
	Default: No filter block
,QUAL=*qualifier*	*qualifier*: RX-type address or address in register (2) - (12).
	Default: No qualifier
QMASK=*qmask keywords*	**Default:** QMASK=NONE
,BITQUAL=*bitqual*	*bitqual*: name of a 32-byte field, hexadecimal numeric value (X'xxx'), or address in register (2) - (12)

Syntax	Description
	Default: 32 bytes of X'00'.
,BITCOMPARE=SUBSET	**Default:** BITCOMPARE=SUBSET
,BITCOMPARE=INTERSECT	
,BITCOMPARE=EQUAL	
,SRBEXIT=*exitrtn addr*	*exitrtn addr*: A-type address, or address in register (2) - (12).
,EXIT=*exitrtn addr*	*exitrtn addr*: A-type address or address in register (2) - (12).
,PARM=*parm addr*	*parm addr*: A-type address, or address in register (2) - (12).
,PARM=*parm data*	*parm data*: a fullword of data
,EOT=NO	**Default:** EOT=NO.
,EOT=YES	
,EOM=NO	**Default:** EOM=NO.
,EOM=YES	
,PLISTVER=2	**Default:** Version implied by keywords
,PLISTVER=3	
,PLISTVER=MAX	
,RELATED=(*value*)	*value*: Any text.
,XSYS=NO	**Default:** XSYS=NO.
,XSYS=YES	

Parameters

The parameters are explained as follows:

ACTION=LISTEN
A required parameter that specifies that you want to listen for a specific system event.

,CODE=*event code*
A required parameter that specifies the system event about which the caller wants to be notified. The *event code* can be any of the decimal codes listed in Table 1.

,DTOKEN=*dtoken*
Specifies a 4-byte output field into which the event notification facility (ENF) returns a token to identify the request. To explicitly delete the listen request in the future, you must code this parameter.

,DISABLE=NO
,DISABLE=YES
Indicates if the listen exit should be disabled for future calls in the case where ENF enters recovery processing because of an abend or other error in the listen exit. The default is DISABLE=YES.

,ESTBNME=*estab name*
Specifies the name of the establisher of the listener user exit routine. The name can be 1 to 8 alphanumeric characters. This optional parameter can be helpful for diagnostic purposes. If you specify ESTBNME, you must also specify EXITNME.

,EXITNME=*exitrtn name*
Specifies the name of the listener user exit routine to receive control when the requested event occurs. The name can be 1 to 8 alphanumeric characters. This optional parameter can be helpful for diagnostic purposes. If you specify EXITNME, you must also specify ESTBNME.

,MASEXIT=NO
,MASEXIT=YES
Specifies whether multiple address spaces can use the same listen exit. MASEXIT=NO, the default, specifies that only one address space can use a particular listen exit. MASEXIT=YES specifies that other address spaces can use the same listen exit.

Note: If you specify the MASEXIT parameter, you cannot also specify the SRBEXIT parameter.

,FLTRBLK=*filter block addr*
Specifies the address of an ENF Listener Filter block. This can only be specified for ENF codes that support it. See Table 1 for the ENF codes that support filter blocks. The filter data is specific to the signal code. See the ENF Codes And Meanings section of the *z/OS MVS Programming: Authorized Assembler Services Guide* to find the macro mapping name which would contain the filter block data name of the mapping macro that maps the ENF Listener Filter block for the specific ENF code.. The storage that contains the filter block data can be released following the ENFREQ LISTEN request.

,QUAL=*qualifier*
Specifies a four-byte value. The four-byte value, called a qualifier, further defines the event. The qualifiers that are valid depend on the system event for which you are listening. Table 1 lists the meaning of the valid QUAL values for each event.

To use this keyword, set QUAL equal to a qualifier that is listed in Table 1 for your event code. The mapping macro that defines symbolics possible for the qualifier also appears in Table 1.

The listener user exit receives control only when a system event occurs that matches the characteristics specified by the QMASK bytes of the hexadecimal value. For example, if QMASK=BYTE1, the listener user exit routine receives control when an event with characteristics described by the first byte in the qualifier occurs, and ENF ignores information in bytes 2 through 4 because QMASK=BYTE1. Note that a QMASK value of NONE (the default) results in the signal being delivered to the listener regardless of the QUAL value provided on the signal. It is recommended that when QUAL is specified, a value other than "NONE'' for QMASK should also be specified.

If your listen request also specifies the BITQUAL keyword, the listen exit receives control only when the system event also matches the characteristics described by the bit-mapped qualifier and bit-wise comparison operator you specify. The system event is only delivered if your listen request also specifies the FLTRBLK keyword. If your listen request also specifies the FLTRBLK keyword, the listen exit receives control only when those filters are also passed. See the BITQUAL and BITCOMPARE parameter descriptions.

,QMASK=*qmask keywords*
Specifies which bytes of the four-byte qualifier ENF uses to further define the event. The listener user exit receives control only when a system event occurs that matches the characteristics specified by the QMASK bytes of the QUAL field.

To specify the bytes of the qualifier that ENF is to use, code any combination of the following keywords separated by commas. If you specify ALL or NONE, ENF ignores all other QMASK keywords. If you do not specify any QMASK keywords, the default is NONE. Because a QMASK value of NONE negates the existence of the QUAL parameter, it is recommended that when specifying QUAL, a value other than NONE should be specified for QMASK. The qualifier that is specified by QUAL is ignored.

BYTE1
First byte

BYTE2
Second byte

BYTE3
Third byte

BYTE4
Fourth byte

ALL
All four bytes

NONE
No bytes are used.

,BITQUAL=*bitqual*
Specifies a 32-byte field, a hexadecimal constant, or a register containing the address of a 32-byte field containing a bit-mapped qualifier that further defines the event. The qualifiers that are valid depend on the system event for which you are listening.

To use this keyword, set BITQUAL as described in Table 1. The figure also lists the mapping macro that defines symbolic values for the qualifier, if any. If you do not specify BITQUAL, the system responds as if you had provided a bit-mapped qualifier with all bits set to zero.

The listen exit receives control only when a system event occurs that matches the characteristics specified by the bit-mapped qualifier and the comparison operation specified by the BITCOMPARE parameter. For example, if BITCOMPARE=INTERSECT, the listener user exit receives control when an event with characteristics represented by any of the bits that are set to '1' in the bit-mapped qualifier occurs.

If your listen request also specifies the QUAL keyword, the listen exit receives control only when the system event also matches the characteristics specified by the QMASK bytes of the QUAL field. The system event is only delivered if your listen request also specifies the FLTRBLK keyword. If your listen request also specifies the FLTRBLK keyword, the listen exit receives control only when those filters are also passed. See the description of the QUAL and QMASK keywords in this information.

,BITCOMPARE=SUBSET
,BITCOMPARE=INTERSECT
,BITCOMPARE=EQUAL
Specifies the comparison operation ENF uses to interpret the bit-mapped qualifier specified with the BITQUAL parameter. In the examples provided with the following parameter descriptions, only 8 of the 256 bits in the bit-mapped qualifier are shown.

- SUBSET, the default, specifies that ENF is to pass control to the listener user exit when an event with characteristics represented by all of the bits that are set to '1' in the bit-mapped qualifier occurs.

  For example, if BITQUAL=X'A0...' (B'10100000...') and BITCOMPARE=SUBSET, ENF passes control to the listener user exit for a system event described by any of the following bit patterns:

  – B'10100000...'
  – B'11100000...'
  – B'10111111...'

  Note: The above list is not exhaustive.

  In all these cases, the characteristics described by the BITQUAL parameter are a subset of the event's characteristics. That is, every bit set to '1' in the bit-mapped qualifier is also set to '1' in the bit pattern describing the system event.
- INTERSECT specifies that ENF is to pass control to the listener user exit when an event with characteristics represented by any of the bits that are set to '1' in the bit-mapped qualifier occurs.

For example, if BITQUAL=X'A0...' (B'10100000...') and BITCOMPARE=INTERSECT, ENF passes control to the listener user exit for a system event described by any of the following bit patterns:

- B'10000000...'
- B'00100000...'
- B'10111111...'

Note: This list is not exhaustive.

In all these cases, the intersection of the characteristics described by the BITQUAL parameter and the characteristics of the event is non-null. At least one bit set to '1' in the bit-mapped qualifier is also set to '1' in the bit pattern describing the system event.

- EQUAL specifies that ENF is to pass control to the listener user exit when an event with characteristics exactly represented by the bit-mapped qualifier occurs.

  For example, if BITQUAL=X'A0...' (B'10100000...') and BITCOMPARE=EQUAL, ENF passes control to the listener user exit only for a system event described by the bit pattern B'10100000...'. In this case, the characteristics described by the BITQUAL parameter exactly match the characteristics of the system event that has occurred, and the bit-mapped qualifier exactly matches the bit pattern describing the system event.

To specify that ENF is not to consider the bit-mapped qualifier when determining whether the listener user exit is to receive control, do one of the following:

- Omit both the BITQUAL and the BITCOMPARE parameters, or
- Specify BITQUAL=0 and BITCOMPARE=SUBSET

,SRBEXIT=*exitrtn addr*
Specifies the address of a listener user exit routine that receives control when the requested event occurs. The specified routine receives control in SRB mode in the address space that issued the listen request. SRBEXIT is valid only with certain event codes. The combination of EOM=NO and EOT=YES is not allowed with SRBEXIT. Do not let EOM default to NO.

If you specify SRBEXIT, you cannot also specify EXIT. See 'Coding the Listener User Exit Routine' in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about SRBEXIT environment.

,EXIT=*exitrtn addr*
Specifies the address of the listener user exit routine that receives control when the requested system event occurs. If you want this listener user exit routine to run in 31-bit mode, you must turn on the high order bit of the exit routine's address; otherwise, the exit gets control in 24-bit mode.

If you specify EXIT, you cannot also specify SRBEXIT. See Exit Routine Environment in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about EXIT Environment.

,PARM=*parm addr*
Specifies the address of a parameter list that the ENF listener can use to pass parameters to the listener user exit routine. This address is stored into the third word of a six-word data structure pointed to by register 1 on entry to the listener user exit routine.

The fifth word of the six-word data structure is the address of the area mapped by the IEFENFSG macro. If the signal for which your listen exit is invoked originated on another system, the area mapped IEFENFSG identifies the target system. The sixth word of the data structure is reserved for possible ALET-qualification of the address mapped by the IEFENFSG macro.

You can specify either PARM=*parm addr* or PARM=*parm data*.

,PARM=*parm data*
Specifies a fullword of data that is stored into the third word of a six-word data structure pointed to by register 1 on entry to the listener user exit routine. Use PARM to pass data to either a standard or an SRB listener user exit routine.

The six-word data structure pointed to by register 1 on entry to the listener user exit routine:

- Address of parameter list supplied by the system for this event code
- Fullword of zeros
- Fullword of data specified by the PARM parameter of the listen request that established the listen exit.
- Fullword of zeros
- Address of a parameter list mapped by the IEFENFSG macro
- Fullword of zeros

,EOT=YES
Specifies that, if the task that issued the listen request ends, ENF no longer passes control to the listener user exit routine when the specified event occurs. EOT=YES is valid only in TCB mode with EOM=YES.

,EOT=NO
Specifies that, if the task that issued the listen request ends, ENF continues to pass control to the listener user exit routine when the specified event occurs. EOT=NO is the default.

,EOM=YES
Specifies that, if the address space that issued the listen request ends, ENF no longer passes control to the listener user exit routine when the specified event occurs.

,EOM=NO
Specifies that, if the address space that issued the listen request ends, ENF continues to pass control to the listener user exit routine when the specified event occurs. EOM=NO is valid only in TCB mode with EOT=NO. EOT=NO is the default. If you specify SRBEXIT, do not let EOM default to NO.

,XSYS=NO
,XSYS=YES
Specifies whether this listen exit is to receive signals originating from other systems in the sysplex. XSYS=NO, the default, specifies that the listen exit is to receive only signals originating from the local system. XSYS=YES specifies that the listen exit is to receive signals from other systems in the sysplex as well those originating locally. XSYS=YES is valid only for those event codes that are defined to ENF (on the system where the listen request is established), as capable of cross-system notification. For more information about listening for system events, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

,PLISTVER=2
,PLISTVER=3
,PLISTVER=MAX
Specifies the version of the parameter list to be generated by ENFREQ. Note that MAX may be specified instead of a number, and the parameter list is of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that ensures that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

The parameter list field that identifies the version number of the macro is only set when the standard or list form is used, or when PLISTVER is explicitly specified. Be sure that the resulting parameter list version number covers all the keys that you use.

The following listen request keywords require the version 3 (or higher) parameter list:

- BITQUAL
- BITCOMPARE
- FLTRBLK
- MASEXIT
- XSYS

,RELATED=*(value)*
An optional parameter that specifies information used to self-document macros by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and can be any values.

ENF event codes and meanings

The following characteristics vary depending on the event for which you are listening.

Event code
Identifies the event.

Qualifier
Further defines the specific event for which you would like to listen.

Parameter list
Passes information about the event to the listener user exit.

Exit type
Specifies the type of the listener user exit routine, which can be either EXIT or SRBEXIT.

Cross-system capable
Specifies whether the exit is to receive signals from other systems in the sysplex

Table 4. ENF macro event codes

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
19	ICSF issues this ENF event when it completes initialization, when it terminates, or when one or more master keys have changed. See the parameter list description for how to identify which event has occurred.		A 1 word parameter list is passed to the user exit. **Byte** **Contents** **1** Only valid when byte 3-4 contains 0004 **Bit** **Meaning** **0** DES Master key (MK) has changed **1** AES MK has changed **2** RSA MK has changed **3** ECC MK has changed **4** P11 MK has changed **5** RCS MK has changed **2** Unused **3-4** Event being signalled **Value** **Meaning** **0002** ICSF services are available **0003** ICSF services are unavailable **0004** One or master keys have changed. See byte 1.	EXIT or SRBEXIT / YES

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
20	**Notes:** 1. The input save area and the information area (registers 13 and 1) point to areas above 16M. When specifying ENFREQ REQUEST=LISTEN, make sure that the exit routine (EXIT keyword) gets control in AMODE 31. 2. This exit only gets control in task mode in ASID 1. 3. Event 20 "listen" exits should avoid issuing dynamic allocation (SVC99) calls.	The defined QUAL values are: **Qualifier** **Information type** **x'80000004'** System information changed. Any program using the data returned by the CSRSI service should obtain the updated data.	Mapped by SIV1V2V3 DSECT within macro CSRSIIDF. This area contains the current information that would be returned by the CSRSI service when all data is requested (a request type of CSRSI_TYPE_ V1CPC_Machine plus CSRSI_TYPE_ V2CPC_LPAR plus CSRSI_TYPE_ V3CPC_VM), with the exception of the fields whose names begin with “SI00PCCA”. If the SI00PCCA*xxx* fields are needed, the CSRSI service can be called. The SIV1V2V3 area is in 31-bit storage.	EXIT / NO
23	The system or an operator varied a device online. An operator can vary a device online by using the VARY command. For more information about the VARY command, see *z/OS MVS System Commands*.	Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBTBYT2 Fourth byte = UCBTBYT1	Mapped by IEFEVARY	EXIT / NO
24	The system or an operator varied a device offline. An operator can vary a device offline by using the VARY command. For more information about the VARY command, see *z/OS MVS System Commands*.	Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBTBYT2 Fourth byte = UCBTBYT1	Mapped by IEFEVARY	EXIT / NO
25	The system or an operator unloaded a DASD or tape volume. An operator can unload a DASD volume by issuing the VARY command. For more information about the VARY command, see *z/OS MVS System Commands*.	Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBTBYT2 Fourth byte = UCBTBYT1	Mapped by IEZEUNLD	EXIT / NO
28	A dynamic device reconfiguration (DDR) swap occurred. A DDR swap moves or swaps a demountable volume from a failed device to another available device. For information about the SWAP command, which enables an operator to perform a DDR swap, see *z/OS MVS System Commands*.	None	8-byte parameter list. The first four bytes contain the address of the UCB for the device that was the source of the swap event. The second four bytes contain the address of the UCB for the device that was the target of the swap event.	EXIT / NO
29	The system or an operator placed a device in pending offline status. An operator can place a device in offline status by issuing the VARY command. For more information about this command, see *z/OS MVS System Commands*.	Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBTBYT2 Fourth byte = UCBTBYT1	Mapped by IEFEVARY	EXIT / NO
30	The system or an operator placed a volume online so that it would be available for system use. An operator can place a volume online by issuing the VARY command. For more information about this command, see *z/OS MVS System Commands*.	Corresponds to the UCBTYP field in the UCB data area. The bytes in the qualifier correspond to the bytes in UCBTYP as follows: First byte = UCBDVCLS Second byte = UCBUNTYP Third byte = UCBTBYT2 Fourth byte = UCBTBYT1	Mapped by IEFEVARY	EXIT / NO
31	A configuration change that involves deleting a device or deleting a path to a device was requested or was rejected.	None	Mapped by IOSDDCCD	EXIT / NO
32	A configuration change was successful.	None	Mapped by IOSDDCCD	EXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
33	One of the following changes to the hardware configuration of a device occurred: • A device is added or deleted from the hardware configuration definition or a device is attached or detached with the VM ATTACH or DETACH command. The I/O subchannel corresponding to the device's UCB is connected or disconnected. • A device is made available because the channel path to the device is reestablished. • The description of a device is added, deleted, or changed. The self-description information is stored in a configuration data record (CDR). A change to a CDR is always a delete followed by an add. Use timestamps to determine the correct sequence. • The HyperPAV mode of operation for a logical control unit is changed. • A change in state has occurred for a PCIe device. • A device requires monitoring. • The supported facilities or capabilities of one or more devices have changed.	**BYTE 1** Device class (Byte 3 from UCBTYP) or zero when DACHQN=X'000A' and DACHTYPE='FACL' **BYTE 2** Reserved **BYTES 3-4** Qualifier number Each qualifier number designates a type of change, such as I/O subchannel change, device available, a configuration data record (CDR) change, or a HyperPav mode change. Along with each qualifier number is a qualifier number-dependent mapping in the IOSDDACH mapping macro, which designates fields specific to the type of change. The following ENF signal 33 subtypes are issued for PAV-alias devices: • DACHIO • DACHIORA • DACHCCDR • DACHPAV For each of these subtypes, if the signal applies to a device in the alternate subchannel set, the issuer will fill in a new field in the DACH subtype for the subchannel set identifier and change the subtype according to the information in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). ENF 33 is issued once for the logical control unit when its HyperPav mode is changed. The following ENF 33 subtype fields are updated to uniquely identify this event: • DACHDEVC='CU' • DACHTRAN='TRAN' • DACHQN=X'0008' • DACH_TRAN_CU=control unit that is changing • DACH_TRAN_MODE=target mode of operation For specific field definitions, see the IOSDDACH macro in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). The ENF signal 33 subtype DACHPCIE (X'0009') is issued for the change in state of a PCIe device. For this subtype, the following value is set in the DACHTYPE field by the issuer of the signal: DACHTYPEPCIE ('PCIE'). For this subtype, the DACHQN field is set to X'0009' (PCIE device event).	Mapped by IOSDDACH	EXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
33 (cont.)		For this subtype, the following data is supplied in the DACHQUALD field by the issuer of the signal: • DACH_PCIE_PFID (4 bytes): The PFID of PCIe device involved in the event. • DACH_PCIE_DEVID (2 bytes): The device ID of PCIe device involved in the event. • DACH_PCIE_VENDID (2 bytes): The vendor ID of PCIe device involved in the event. • DACH_PCIE_EVENT (1 byte): The device event code: 1 = The device is going online. 2 = The device is going offline. ENF signal 33 subtype DACHMONC (DACHTYPE = DACHMONC) is issued when a change in device monitoring is requested. This signal may be generated for secondary devices monitored for HyperSwap® configurations for which I/O operations may begin to be started. This signal may also be received when devices that were previously identified by ENF 33 subtype DACHMONC no longer require monitoring. Programs such as RMF may choose to monitor this ENF 33 signal to know when to begin collecting data for these devices which might otherwise see only insignificant amounts of I/O activity. For the DACHMONC subtype: • For a device that requires monitoring, the following fields are set: **DACH_IO_QUAL** Set to DACH_IO_QUAL_MONC_ON **DACH_IO_DEVN** Device number **DACH_IO_SSID** Subchannel set identifier **DACH_IO_DTYP** Contents of the UCBTYP field from the UCB **DACHUCBC** Device class **DACHQN** Either DACHIO or DACHIO_AS Each device receives a separate signal to begin monitoring. When monitoring is requested, the UCBCMONR bit is set on in the UCB. • When one or more devices no longer require monitoring, a single ENF 33 DACHTYPE = DACHMONC, DACHQN = DACHIO signal is given with DACH_IO_QUAL = DACH_IO_QUAL_MONC_OFF. This is typically done after a HyperSwap occurs or when a configuration is purged from the HyperSwap manager. When this signal is received, a UCBSCAN can be done to detect devices for which the UCBCMONR bit is no longer set on. DACH_IO_DEVN and DACH_IO_SSID are not used for this signal.		

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
33 (cont.)		ENF signal 33 subtype DACHFACL (X'000A') is issued when the supported facilities or capabilities of one or more devices have changed. For example, the zHyperLink facility is enabled or disabled for one or more devices. The following ENF 33 subtype fields are updated to uniquely identify this event: • DACHQN=X'000A' (DACHFACL) • DACHTYPE='FACL' (DACHTYPEFACL) The device class is zero in the ENF qualifier and DACHUCBC field. The qualifier dependent data field (DACHQUALD) is not used for this event. Instead, a list of affected devices is provided at the end of the DACH parameter list (DACHEND field). The list consists of the following: • A header mapped by DACH_FACL_Fields. The header contains the number and length of each device entry that follows. • One or more device entries mapped by DACH_FACL_DevEntry. An individual device is represented by one device entry, and a range of devices is represented by a pair of device entries. The type of facility or capability that changed is not provided. The program should use a service such as UCBINFO or examine the UCB to determine which facilities or capabilities have changed.		
35	One of the following XES or XCF events has occurred: • New coupling facility resources have become available on this system. Requests to connect with IXLCONN that previously failed might now succeed because of this new coupling facility resource. • A specific structure has become available for use. Requests to connect to the structure with IXLCONN that previously failed might now succeed because of this new coupling facility resource. • A system has joined the sysplex. The system name and ID are presented to the user. • A system has been partitioned from the sysplex. The system name and ID are presented to the user. • A CF definition with a SITE specified has been added or an existing CF SITE specification has changed. Note that the listener user exit routine for event code 35 can run in SRB mode.	None	Mapped by IXCYENF	EXIT or SRBEXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
36	ENF passes a parameter list containing the logrec record data to the listener user exit routine. • When SETLOGRC DATASET or LOGSTREAM is active, the ENF signals are issued until the logrec medium (for instance, the data set) is filled. ENF signals will resume once the medium is available. • When SETLOGRC IGNORE is active, no ENF signals are issued. • When SETLOGRC ENF is active, the ENF signals are unabated. For details about the contents of the parameter list, see IFBENF36 in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). Additional considerations for listeners of this code include the following: • The mapping does not indicate whether an IBM or non-IBM program caused the record to be written to logrec. • ENF does not suppress duplicate ENF signals sent to the listener. The listener must be aware of instances where a program loop causes the same software record to be recorded in logrec multiple times, thus causing ENF to issue duplicate signals. • ENF does not filter software records based on any criteria including ABEND codes.	The specific logrec record type value is used as the qualifier for each ENF event code 36 signal. Note that no signal is issued for record types X'9*x*'.	Mapped by IFBENF36	EXIT or SRBEXIT / NO
37	One of the following SMF accounting-related events occurred: • SMF was initialized • SMF ended • SMF INTVAL parameter changed • SMF SYNCVAL parameter changed • SMF interval expired • SMF interval sync processing disabled • SMF event driven interval occurred For information about these accounting-related events, see *z/OS MVS System Management Facilities (SMF)*.	**ENF37Q00** SMF address space was initialized. **ENF37Q01** SMF address space ended. **ENF37Q02** SMF INTVAL parameter changed. **ENF37Q03** SMF SYNCVAL parameter changed. **ENF37Q04** SMF interval expired. **ENF37Q05** SMF interval sync processing disabled. **ENF37Q06** SMF event driven interval occurred.	Mapped by IFAENF37	EXIT / NO
38	One of the following automatic restart manager events occurred: • A job or task started or was restarted, and has registered or reregistered as an element of the automatic restart manager. • An element notified the system that it is ready to accept work. • An element has deregistered with the automatic restart manager. • This system has acquired (or regained) access to the automatic restart management couple data set. Batch jobs and started tasks may now register as elements of the automatic restart manager. • An element has been deregistered with the automatic restart manager.	None	Mapped by IXCYAREN	SRBEXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
40	A JES2 subsystem either completed initialization or ended normally. (Note that ENF code 40 does not reflect situations in which JES2 abends.) ENF passes to the listener user exit routine a parameter list that identifies the JES2 subsystem. For details about the contents of the parameter list, see IEFENF40 in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).	**ENF40_INIT** A JES completed initialization. **ENF40_TERM** A JES ended normally	Mapped by IEFENF40	EXIT / NO
41	A workload management (WLM) event occurred. The following qualifiers for ENF code 41 are provided: **BYTE 1** **1** Policy change was initiated. **2** Policy change completed. **3** Policy change failed. **BYTE 2** Reserved. **BYTE 3** **1** Workload activity reporting failed and has begun recovery. **2** Workload activity reporting recovery was successful. **3** Workload activity reporting recovery was not successful. **BYTE 4** **1** WLM service definition was successfully installed.	**WLMENF11** A VARY WLM,POLICY command was issued. **WLMENF12** A VARY WLM,POLICY command completed. **WLMENF13** A VARY WLM,POLICY command failed. The new policy could not be activated on this system. **WLMENF31** WLM workload activity reporting failed and has begun recovery. **WLMENF32** WLM workload activity reporting recovery was successful. **WLMENF33** Workload activity reporting recovery was unsuccessful. **WLMENF41** Service definition was successfully installed.	Mapped by IWMRENF1	EXIT / NO
43	A new copy of workload management sampled address space information is available via IWMRQRY. Event code 43 is issued at the end of workload management's sampling interval so a listener can synchronize its sampling interval with workload management's interval.	None	Four byte parameter containing the length of the storage required to hold the information. A listener can pass this length to IWMRQRY in the ANSLEN parameter and save issuing IWMRQRY to determine the length.	EXIT / NO
44	A configuration change involving paths to a coupling facility has occurred.	None	Mapped by IXLYCFSE	EXIT / NO
45	The SMSVSAM server address space has been initialized or reinitialized after a failure. Any subsystem that lost connection to the service provider address space can now reconnect.	None	Mapped by IDAENF45	SRBEXIT / NO
46	z/OS UNIX System Services has been initialized or reinitialized.	None	None	EXIT / NO
47	DAE has detected that the threshold for completed or suppressed dumps, related to a particular symptom string, has been reached.	None	Mapped by ADYENF	EXIT / NO
48	A status change has occurred within system logger. The events issued by ENF 48 are issued to all systems in the sysplex. For a description of using ENF event 48 for system logger, see *z/OS MVS Programming: Authorized Assembler Services Guide*. For a description of the events mapped by the IXGENF macro, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).	None	Mapped by IXGENF	SRBEXIT / YES

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
49	The logrec output recording medium has been changed by the SETLOGRC command.	None	IFBNTASM	SRBEXIT / NO
51	One of the following types of GRS information: • Resource contention information • RNL change effects on user jobs • GRS mode change information Note that the listener user exit routine for event code 51 can run in SRB mode. Event code 51 can generate large numbers of events in short periods of time. The listener user exit routine for event code 51 must handle the volume of events. See *z/OS MVS Programming: Authorized Assembler Services Guide* for a description of system services to avoid when writing listener user exits. Supports Filter Block (FLKBLOCK) listners: Mapped by ISGYELF. The filter block reason codes for EnfReq RC=X'68' is in field ISGYELF_ReasonCode. The mapping also includes constants for the various values of the reason code.	The qualifier (QUAL parameter) has the following format: **BYTE 1** Type of signal information: **x'01'** Contention data **x'02'** RNL changes **x'03'** Mode changes **BYTE 2** Always x'00'. **BYTE 3** Varies with type of signal (value of BYTE1): **x'00'** Normal contention **x'01'** Waitless contention **BYTE 4** Varies with type of signal (value of BYTE1): **x'01'** Local events **x'02'** Global events **x'03'** Recovery events The defined QUAL values are: **Qualifier** **Information type** **x'01000000'** All Normal resource contention (excludes waitless) **x'01000001'** Normal Local resource contention **x'01000002'** Normal Global resource contention **x'01000003'** Normal Contention-related recovery information **x'01000100'** All Waitless resource contention **x'01000101'** Waitless Local resource contention **x'01000102'** Waitless Global resource contention **x'02000001'** User job suspended because of RNL change **x'02000002'** User job resumed following RNL change **x'0300*yyzz*'** GRS mode changes: ***yy*** Old mode ***zz*** New mode Values for *yy* and *zz* are those defined in IHAECVT for the ECVTGMOD field	Contention data: ISGE51CN RNL data: ISGE51RN Mode change data: None	EXIT or SRBEXIT / YES
52	A LNKLST set has been activated. A LNKLST set can be activated at IPL through a PROGxx LNKLST statement, or through SET PROG=xx or SETPROG LNKLST operator commands. For information about PROGxx, see *z/OS MVS Initialization and Tuning Reference*. For commands, see *z/OS MVS System Commands*.	None	Mapped by CSVDLENF	EXIT or SRBEXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
53	• A Sysplex Timer (ETR) configuration change occurred. • A change to the local time offset occurred.	None	8-byte parameter list • Bytes 1 and 2 indicate a configuration change to the Sysplex Timer (ETR). • Byte 3, if non-zero, indicates a change to the local time offset. Possible values are: **0** Sysplex Timer configuration has changed. **1** Local time offset has changed. **2** Leap second offset has changed. **3** Both local time offsets have changed.	EXIT / NO
55	The system resource manager (SRM) has detected a significant MVS image event, which is being signalled. The qualifiers and parameters further define the event.	**ENF55QLF_REAL_SHORTAGE (X'80000000')** Too many fixed frames in storage; issued when IRA400E occurs. **ENF55QLF_REAL_SHORTAGE_RELIEVED (X'40000000')** Pageable storage shortage due to excessive fixed storage relieved; issued when IRA402I occurs. **ENF55QLF_REAL_WARNING (X'20000000')** Pageable storage warning that indicates there are many fixed frames in storage; issued when IRA405I occurs. **ENF55QLF_AUX_CRITICAL_SHORTAGE (X'08000000')** Too many slots allocated in the AUX subsystem. It is a critical shortage and is issued when IRA201E occurs. **ENF55QLF_AUX_SHORTAGE (X'04000000')** Too many slots allocated in the AUX subsystem. It is issued when IRA200E occurs. **ENF55QLF_AUX_SHORTAGE_RELIEVED (X'02000000')** AUX Storage shortage due to excessive slots relieved. It is issued when IRA202I occurs. **ENF55QLF_AUX_WARNING (X'01000000')** AUX Storage usage warning that indicates there are many slots allocated in the AUX subsystem. It is issued when IRA205I occurs. **ENF55QLF_SCM_HIGH_USAGE (X'00040000')** High usage of storage-class memory (SCM). Issued when IRA250I occurs.	Mapped by IRAENF55	EXIT or SRBEXIT / NO
55 (cont.)		**ENF55QLF_SCM_HIGH_USAGE_RELIEVED (X'00020000')** High usage of storage-class memory (SCM) relieved. Issued when IRA252I occurs. **ENF55QLF_AFQ_SHORTAGE (X'00008000')** Available frame queue shortage. Not enough frames on the available frame queue. **ENF55QLF_AFQ_SHORTAGE_RELIEVED (X'00004000')** Available frame queue shortage relieved. **ENF55QLF_PREF_SHORTAGE (X'00002000')** Preferred frame queue shortage. Not enough frames on the preferred frame queue. **Note:** This preferred storage shortage indicator is an informational notification for applications that are able to change their storage allocation type. In case of a preferred storage shortage, the application should request non-preferred storage (if possible) instead of preferred storage. No action is taken by SRM to address this shortage. **ENF55QLF_PREF_SHORTAGE_RELIEVED (X'00001000')** Preferred frame queue shortage relieved.		

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
56	Workload management has changed an attribute of a job.	**WLMENF56_QUAL_RESET** A job was reset using the RESET system command or IWMRESET macro. **WLMENF56_QUAL_ ENCLAVERESET** An enclave has been successfully reset via the IWMERES service.	Mapped by IWMRENF2	EXIT / NO
57	The state of a workload management scheduling environment has been altered.	**WLMENF57_NORMAL_ SCHENV_CHANGE** The state of a scheduling environment has changed due to a F WLM,RESOURCE command or IWMSESET macro. **WLMENF57_RECOVERY_ SCHENV_CHANGE** The state of a scheduling environment has changed due to workload management recovery processing.	Mapped by IWMRENF57	EXIT / NO
58	The state of a SYSOUT data set has changed. The state of a SYSOUT data set changes when it is either dynamically allocated using the DALRTCTK text unit, or when the SYSOUT application program interface (SAPI) disposition bit is set. For more information, see the "Listening for Events" section of the "JES Client/ Server Print Interface" chapter in *z/OS JES Application Programming*.	**ENF58_Q_PURGE** The data set was purged. **ENF58_Q_SELECT** The data set was selected. **ENF58_Q_DESELECT_ PROCESSED** The data set was processed. **ENF58_Q_DESELECT_ NOT_PROCESSED** The data set is no longer selected, disposition was not updated. **ENF58_Q_DESELECT_ NOT_PROCESSED_HELD** The data set is no longer selected, disposition was not updated, and data set is held. **ENF58_Q_DESELECT_ ERROR** An error resulting in a system level hold occurred. **ENF58_Q_EOD_OK** End of data set notification occurred — successful. **ENF58_Q_EOD_ERROR** End of data set notification occurred — unsuccessful. **ENF58_Q_JOB_CHANGE** A job status change occurred. **ENF58_Q_TOKEN_ CHANGE** The client token has changed. **ENF58_Q_INSTANCE** Addition instance of data set created. **ENF58_Q_GRP_SELECT** Data set group select. **ENF58_Q_GRP_DESELECT** Data set group deselect.	Mapped by IAZENF58	EXIT / YES
60	A TRACE TT command has been accepted.	ENF60_QUAL	Mapped by ITZENF60	EXIT / NO
61	The capacity of the MVS image or CEC has changed.	WLMENF61_CAPACITY_ CHANGE	Mapped by IWMENF61	EXIT / NO
62	A RACF® SETROPTS RACLIST command has affected in-storage profiles used for authorization requests in a class designated as SIGNAL=YES or SIGNAL(YES) in the RACF class descriptor table. The class affected is in the parameter list in field IRR_ENFCLASS.	The qualifier (QUAL) has the following format: **BYTE 1** **X'80'** SETROPTS RACLIST has taken place. **BYTE 2** **X'80'** SETROPTS RACLIST REFRESH has taken place. **BYTE 3** **X'80'** SETROPTS NORACLIST has taken place.	Mapped by IRRPENFP in SYS1.MACLIB.	EXIT or SRBEXIT/ NO
63	A permanent error was detected on a HyperSwap capable device.	None	Mapped by IOSDE63R	EXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
64	One of the following events occurred: • The capacity of a storage volume has changed. • The VTOC or INDEX of a direct access volume has been extended or moved to a new location. • The VTOC index of a storage volume has been built. The index indicates a direct access volume has changed from an OS format VTOC (OSVTOC) to an indexed format VTOC (IXVTOC). • The content of the volume has changed due to a full volume copy or restore operation. • IBM DS8000® recovery scenario occurs either on primary or secondary PPRC disk subsystem. IBM DS8000 signals z/OS on all paths via the Storage Controller Health Message attention status. • One or more devices in the logical subsystem has a PPRC state change. • One or more devices in the logical subsystem has a multi-target PPRC state change. • A full volume FlashCopy® relationship has been established. • Safeguarded Copy Backup Volume has reached a warning threshold. • Safeguarded Copy Backup Volume space has become completely exhausted. • Safeguarded Copy Backup Volume physical space has been relieved. • Safeguarded Copy Backup Volume Consistency Group was removed to reclaim space for the volume.	The qualifier (QUAL parameter) has the following format: **BYTE 1** Type of signal information: **X'01'** Volume event **X'02'** LSS event **X'03'** Out of Space event **BYTE 2** Varies with event **BYTE 3** Always X'00' **BYTE 4** Varies with event The defined QUAL values are: **Qualifier** **Information type** **X'0100*xxxx*'** Volume events **X'01000001'** DASD volume capacity changed **X'01000002'** VTOC updated (moved or extended) **X'01000003'** VTOC index is built **X'01000010'** Volume transformed. This qualifier indicates that the content of the volume, including system data, such as the VTOC, VTOC INDEX, and VVDS, has changed, and the location of these files may have changed. **X'02*xxxxxx*'** LSS event **X'02010001'** Storage controller health (LSS) event **X'02020001'** Summary (LSS) event - PPRC state change **X'02020002'** Summary (LSS) event - Multi-target PPRC state change **X'0300xxxx'** Out of Space Event **X'03000007'** Safeguarded Copy Backup Volume has reached a warning threshold. **X'03000008'** Safeguarded Copy Backup Volume has become completely exhausted. **X'03000009'** Safeguarded Copy Backup Volume physical space constraint has been relieved. **X'0300000A'** Safeguarded Copy Backup Volume Consistency Group was removed to reclaim space for the volume.	Mapped by IECENF64	EXIT / YES

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
65	System REXX event has occurred.	**X'80000000'** The AXR address space has initialized. AXREXX can be invoked. **X'40000000'** The AXR address space has terminated. Subsequent AXREXX invocations will be rejected. **X'20000000'** The AXR address space has reached its threshold of ACTIVE+WAITING AXREXX requests. No more requests will be accepted until the number of requests drops to an acceptable level. **X'10000000'** The number of ACTIVE+WAITING AXREXX requests has dropped to an acceptable level. AXREXX requests are now being accepted. **X'08000000'** The number of ACTIVE+WAITING AXREXX requests is high and is nearing the level where subsequent requests will be rejected. **X'04000000'** The number of extents in the REXXLIB concatenation exceeds the system limit. See *z/OS DFSMS Using Data Sets* for details. If this condition is detected during System REXX initialization, System REXX terminates; otherwise, no new AXREXX requests will be accepted.	None	EXIT/NO
67	One of the following IBM Health Checker for z/OS events has occurred: • IBM Health Checker for z/OS has become available. • IBM Health Checker for z/OS has terminated and is not available.	The defined BITQUAL values are: **Qualifier** **Information type** **X'80000000'** IBM Health Checker for z/OS is available. Field Enf067_BitQual_Available in the HZSZENF mapping macro. **X'40000000'** IBM Health Checker for z/OS has terminated and is not available. Field Enf067_BitQual_NotAvailable in the HZSZENF mapping macro.	Mapped by HZSZENF	EXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
68	One of the following BCPii events has occurred: • A change in BCPii status has occurred. • A hardware communication error has occurred. • A hardware event has occurred.	The defined QUAL values are: **Qualifier** **Information type** **X'01000001'** BCPii is available. **X'01000002'** BCPii is not available. **X'020100*yy*'** A hardware communication error has occurred and CPC events might have been lost. *yy* denotes the type of error: **01** A temporary error, some events might have been lost. **02** A permanent error, no more events are delivered. **03** Communication to the CPC has been established or reestablished. Event delivery from this CPC will now commence or re-commence. **X'03*xx*00*yy*'** A hardware event has occurred. *xx* denotes the event source: **01** CPC **02** Image *yy* denotes the event. The defined BITQUAL values are: **Qualifier** **Information type** **X'01nnnnnn'** N/A **X'0201nnnn'** Bytes 1-17 CPC name, padded with hexadecimal zeros **X'0301nnnn'** Bytes 1-17, CPC name, padded with hexadecimal zeros **X'0302nnnn'** Bytes 1-17, CPC name, padded with hexadecimal zeros Bytes 18-24, image name, padded with hexadecimal zeros	Mapped by HWICIASM and HWICIC	EXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
68	One of the following BCPii events has occurred: • A change in BCPii status. • A hardware communication error has occurred. • A hardware event has occurred.	Hardware Event Codes: **Code** **Description** **'01' x** A command response has been received. **'02' x** An object status change has occurred. **'03' x** An object name change has occurred. **'04' x** The activation profile of the object has changed. **'05' x** A new object was created. **'06' x** An object was deleted. **'07' x** An object entered or left an exception state. **'08' x** A Console application has started. **'09' x** A Console application has ended. **'0A' x** An operating system message has been received. **'0B' x** A hardware message has been received. **'0C' x** A hardware message has been deleted. **'0D' x** A capacity change event has been received. **'0E' x** A capacity record change has occurred. **'0F' x** A security event has been logged. **'10' x** An image has entered a disabled wait state. **'11' x** A power change event has been received.	Mapped by HWICIASM and HWICIC	EXIT / NO
70	The state of a job (batch, STC or TSU) owned by JES has changed. The job may have been selected for processing, completed processing, changed phase (including changes to execution phase job class), or been purged.	**ENF70_SELECT** Job was selected. **ENF70_DESELECT** Job was processed. **ENF70_CHANGE** Job queued to new phase of processing. **ENF70_PURGE** Job was purged.	Mapped by IAZENF70	EXIT / YES
71	A RACF command has affected a user's group connections which may affect his or her resource authorization. The user affected is in the parameter list in field IRR_ENF2USER. The group affected is in the parameter list in field IRR_ENF2GROUP. Control flags that are used to provide greater granularity for the listeners are in the parameter list in field IRR_ENF2Flags.	The qualifier (QUAL) has the following format: **BYTE 1** **X'80'** CONNECT command **X'40'** REMOVE command **X'20'** ALTUSER REVOKE command **X'10'** DELUSER command **X'08'** DELGROUP command **BYTES 2 - 4** Reserved	Mapped by IRRPENF2 (See *z/OS Security Server RACF Data Areas*)	EXIT or SRBEXIT / YES

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
72	Volume status information for SMS. The listener user exit routine for event code 72 can run in SRB mode. Event code 72 can generate large numbers of events in short periods of time. The listener user exit routine for event code 72 must be able to handle the volume of events.	The qualifier (QUAL) has the following format: **BYTE 1** Type of signal information: **X'01'** Volume status **BYTE 2** Always X'00' **BYTES 3 - 4** Varies with event. The defined QUAL values are: **Qualifier** **Information type** **X'01000001'** ENF72_OVER_THRESHOLD_AM_Y. Volume over threshold when storage group is defined using AM=Y (Auto Migrate, yes).	Volume Status information: IGDE72VL Mapped by IGDENV72	EXIT or SRBEXIT / YES
73	The SETLOAD xx, IEASYM command has completed successfully. The local system's symbol table has been updated.	None	None	EXIT/ NO
78	The state of a job (batch, STC or TSU) owned by JES has changed. The job has completed processing.	ENF78_JOB_NOTIFY	Mapped by IAZENF78	EXIT/YES
79	A RACF command has modified a profile such that a user's authorization to the resources it protects may be affected. • The user affected is in the parameter list in field IRR_ENF3_UserID. • The class in which the modified profile belongs is in the parameter list in field IRR_ENF3_ClassName. • The length of the affected profile name is in the parameter list in field IRR_ENF3_ProfName_Length. • The name of the affected profile is in the parameter list in field IRR_ENF3_ProfName. Control flags that are used to provide greater granularity for the listeners are in the parameter list in field IRR_ENF3_Flags. For the PERMIT RACF command processor, there maybe additional information regarding: • The type of Conditional Access, a numerical value that is in the parameter list in field IRR_ENF3_PERMIT_WHEN_Cond. • The Conditional Access List Entry. The length of the Conditional Access Name and the Conditional Access Name itself is in the parameter list in the fields: IRR_ENF3_CACLName_Length IRR_ENF3_CACLName For the RDEFINE and RALTER RACF command processors, there may be additional information in the ADDMEM and DELMEM lists. The number of elements in the list, the length of the list, and the offset to the list are in the parameter list in the fields: IRR_ENF3_ADDMEML_Member# IRR_ENF3_DELMEML_Member# IRR_ENF3_ADDMEML_Length IRR_ENF3_DELMEML_Length IRR_ENF3_ADDMEML_Offset IRR_ENF3_DELMEML_Offset	The qualifier (IRR_ENF3_QualCode) has the following format: **BYTE 1** **X'80'** PERMIT command **X'40'** RDEFINE command **X'20'** RALTER command **X'10'** DELETE command **BYTES 2 - 4** Reserved	Mapped by IRRPENF3 (See *z/OS Security Server RACF Data Areas*)	EXIT or SRBEXIT / YES

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
80	One of the following z/OS Communication Server events has occurred: • The rpcbind server has initialized. • The rpcbind server is stopping. **Notes:** 1. ENF80_RPC DSECT maps the RPCBIND event. 2. Use the ENF80_RPC_FLAGs to determine if the rpcbind server is initializing or ending. 3. When flag ENF80_RPCINIT is on, RPC applications can register with RPCBIND. 4. When flag ENF80_RPCTERM is on, the rpcbind server is stopping. 5. ENF80_RPC DSECT includes the jobname of the rpcbind server that generated the event.	ENF80_RPC_EVENT	Mapped by EZAENF80	SRBEXIT / NO
	One of the following z/OS sysplex events has occurred: • The TCP/IP stack has joined the sysplex group. • The TCP/IP stack has left the sysplex group. **Notes:** 1. ENF80_SYSPLEX DSECT maps the TCP/IP event. 2. Use the ENF80_SYSPLEX_FLAGS to determine if the TCP/IP stack is joining or leaving the sysplex group. 3. When flag ENF80_SYSPLEX_JOIN is on, the TCP/IP stack is joining the sysplex group. 4. When flag ENF80_SYSPLEX_LEAVE is on, the TCP/IP stack is leaving the sysplex group. 5. ENF80_SYSPLEX DSECT includes the jobname of the TCP/IP stack that generated the event.	ENF80_SYSPLEX_EVENT **ENF80 Qualifier** X'40000000'	Mapped by EZAENF80	EXIT / NO
	82	One of the following IBM Function Registry for z/OS events has occurred: • A function has changed its enabled state. **Notes:** 1. This event will not support filter blocks (FLTRBLK) nor pre-processing exists, but will be allowed to run in SRB, SRB and task, or task modes. 2. The parameter list for the “enablement state change” event, as mapped by macro FXEHENF, will contain information to uniquely identify the function for which the enablement state changed and what the current state after the change is: • Both vendor name and vendor slot number • Both product name and product slot number • Both function name and function slot number • New value of enablement state of the function (enabled or disabled).	The defined BITQUAL values are: **Qualifier** **Information Type** **X’80000000’** A function has changed its enable-ment state. Field Enf082_BitQual_Enablement in the FXEHENF mapping macro.	Mapped by FXEHENF	EXIT / NO

Table 4. ENF macro event codes (continued)

Event code	Description	Qualifier	Parameter list passed to the user exit	Exit type / Cross-system capable
84	ENF signal 84 is sent when there is a Boost event. This signal supports EXIT but not SRBEXIT on the LISTEN request. The signal will be sent only if there had been, or will be, at least one boost active. The intended use is by an application that has noticed that a boost is active and then wants to know when the boosts are no longer active. An ENF 84 signal is sent when a boost starts or ends. An ENF 84 signal is usually sent when a recovery process boost is extended (as can happen if a new recovery process boost is requested when a recovery process boost is already active). An ENF signal is not sent for every extension. It is sent for: • The first extension • Any extension that has a different requestor than the requestor for which the boost is currently being done • Any extension that is more than a minute since the last ENF signal for an extension. The "extend" signal is intended to give applications the opportunity to re-examine the Ecvt_BoostInfo_Expected_EndETOD field which might have changed.	Boost event (field ENF84_Event within the area mapped by IHAENF84)	Mapped by IHAENF84	EXIT / NO

Return codes

When ENFREQ macro returns control to your program, GPR 15 contains a return code.

Table 5. Return Codes for the ENFREQ Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: ENFREQ processing completed successfully. **Action**: None
04	**Meaning**: Program error. An identical LISTEN request already exists. A request is considered a duplicate if its QUAL, QMASK, EXIT, BITQUAL, and BITCOMPARE parameter values are the same as those specified for an existing request. **Action**: None. The request is already established.
0C	**Meaning**: Program error (invalid parameter list) The ENFREQ failed for one of the following reasons: • The length of the parameter list is incorrect. • The specified ACTION code is not valid. • The specified EVENT code is not valid. • The caller specified ACTION=LISTEN, and the EXIT address is zero. • The caller specified ACTION=DELETE, and the DTOKEN field is zero. **Action**: After checking and correcting the program environment parameters, retry the request. If the parameters are correct, check to see if you inadvertently overlaid the control parameter list.
10	**Meaning**: System error. This return code is for IBM diagnostic purposes only. **Action**: Record the return code, and supply it to the appropriate IBM support personnel.
14	**Meaning**: Environmental error. Your program issued the ENFREQ macro before the system initialized ENF. **Action**: Retry the request. If the problem persists, record the return code and supply it to the appropriate IBM support personnel.

Table 5. Return Codes for the ENFREQ Macro (continued)

Hexadecimal Return Code	Meaning and Action
18	**Meaning**: Environmental error. The system cannot obtain storage for your request. **Action**: Rerun your program one or more times. If the problem persists, check with the operator to see if another user in the installation is causing the problem, or if the entire installation is experiencing storage constraint problems.
1C	**Meaning**: Program error. The DTOKEN parameter does not represent any LISTEN request that is currently active. ENF does not perform a DELETE. **Action**: Verify that the DTOKEN on the DELETE request matches the DTOKEN from the LISTEN request. Retry the DELETE request with the correct DTOKEN.
20	**Meaning**: Program error. An abend occurred in the Listen Exit code. **Action**: If a dump was produced for the abend, examine it and correct the programming error.
3C	**Meaning**: Program error. EOT=YES was specified on an ENFREQ listen request while the issuer of the ENFREQ request was running in SRB mode. **Action**: Either specify EOT=NO or delete the EOT keyword from the ENFREQ macro invocation.
46	**Meaning**: Program error. The SRBEXIT keyword was specified on an ENFREQ listen request for an event code that does not allow SRBEXIT. **Action**: Verify that the listen request is for the correct event code. If so, replace the SRBEXIT keyword with the EXIT keyword and ensure that the listen exit resides in common storage.
48	**Meaning**: Program error. The EXIT keyword was specified on an ENFREQ listen request for an event code that does not allow EXIT. **Action**: Verify that the listen request is for the correct event code. If so, replace the EXIT keyword with the SRBEXIT keyword.
4A	**Meaning**: Program error. The keyword combination of EOT=YES and EOM=NO was specified on an ENFREQ listen request. This combination is incorrect. **Action**: Change the EOM specification to YES or the EOT specification to NO.
4C	**Meaning**: Program error. EOM=NO and SRBEXIT were specified on an ENFREQ listen request. This combination is incorrect. **Action**: Change the EOM specification to YES or do not use SRBEXIT.
4E	**Meaning**: Program error. An ENF request specified XSYS=YES for an event code that does not support sysplex-wide notification. **Action**: Verify that the ENF request is for the correct event code. If so, specify XSYS=NO (or allow the XSYS parameter to default to XSYS=NO).
50	**Meaning**: System error. Sysplex-wide notification is not available, because of a system initialization problem. ENF listeners will receive notifications originating from only the system where the listen exit was established. **Action**: Report the problem to the operator and the system programmer. The cross-system signalling capability will remain unavailable until the next system IPL.
52	**Meaning**: Program error. Sysplex-wide notification services were requested for an action type other than listen. **Action**: Verify that your program is not overwriting the parameter list, and that the execute form of the macro correctly addresses the parameter list.
54	**Meaning**: Program error. An ENF request specified invalid comparison instructions for the bit-mapped qualifier. **Action**: Verify that your program is not overwriting the parameter list, and that the execute form of the macro correctly addresses the parameter list.
60	**Meaning**: Program error. An ENF request specified FLTRBLK for an event code that does not support listener filter blocks. **Action**: Verify that the ENF request is for the correct event code. If so, do not specify FLTRBLK.

Table 5. Return Codes for the ENFREQ Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
64	**Meaning**: Program error. An ENF request specified FLTRBLK. It was specified for an event code that does support listener filter blocks, but the block was not accessible by the owner of that particular event code. **Action**: Ensure that the event-specific listener filter block occupies accessible storage of sufficient length.
68	**Meaning**: Program error. An ENF request specified FLTRBLK. It was specified for an event code that does support listener filter blocks, and the block was accessible by the owner of that particular event code, but the filter parameters are incorrect. **Action**: Check the parameters specified in the FLTRBLK. If the event-specific mapping includes a reason code, use its value to assist with the problem determination.

Example 1

Set up and load into common storage the SMFLST00 listener user exit routine, which gains control only if the qualifier equals ENF37Q00.

Note that the qualifiers are declared in the IFAENF37 mapping macro. The ENFREQ macro specifies QMASK=ALL which requests that all four bytes of the qualifier mask are used in the qualifier comparison.

```
* Load ENF Listen Exit (SMFLST00) into common storage and save address.
* SMFL00@ contains the address of the listener user exit routine that
* resides in common storage
         .
         .
         ST    R00,SMFL00@
*
* Issue LISTEN Request for SMF Event Code (Qualifier ENF37Q00)
         L     R02,SMFL00@
         ENFREQ ACTION=LISTEN,    -- Function                         +
               CODE=ENFC37,       -- Event Code                       +
               EXIT=(R02),        -- Exit Address                     +
               QUAL=ENF37Q00,     -- Qualifier Value                  +
               QMASK=ALL,         -- Qualifier Mask (Full Word)       +
               ESTBNME=THISMOD,   -- Establisher Name                 +
               EXITNME=SMFLST00,  -- Exit Name                        +
               DTOKEN=SMFL00T     -- Returned Token Field
*
* Check the return code from ENFREQ - if not zero issue message
*
* - Local variables
DATAAREA DSECT
SMFL00@  DS    A
SMFL00T  DS    F
ENFPTR   DS    A
*
* - Local constants
SMFLSTEN CSECT
         DS    0F
ENFC37   EQU   37
THISMOD  DC    CL8'SMFLSTEN'
SMFLST00 DC    CL8'SMFLST00'
*
* - External control blocks
         CVT   DSECT=YES
         IEFENFCT
SMFLSTEN CSECT
         IFAENF37
DATAAREA DSECT
         IEFENFPM
LENODATA EQU   *-DATAAREA
*
```

Note that the IFAENF37 macro includes the following declarations:

```
&SYSECT  CSECT               Control Section for Constants
ENF37Q00 DC    X'80000000'    SMF Active
ENF37Q01 DC    X'40000000'    SMF Terminated
```

Example 2

Set up and load into storage the ENFLST01 listener user exit routine. This listener user exit routine receives a parameter from the ENF listener when the specified event occurs. The listener user exit runs in the address space of the listener and is deleted when the address space that issued the listen request ends.

```
* Load ENF Listen Exit (ENFLST01) into storage and save address.
* ENFL01@ contains the address of the listener user exit routine.
*
* Issue LISTEN Request for Event Code 35
         L     R02,ENFL01@
         ENFREQ ACTION=LISTEN,    -- Function                        +
               CODE=ENFC35,       -- Event Code                      +
               SRBEXIT=(R02),     -- Exit Address                    +
               PARM=LPARM,        -- Parameter                       +
               EOT=YES,           -- End-of-task delete indicator    +
               EOM=YES,           -- End-of-memory delete indicator  +
               ESTBNME=THISMOD,   -- Establisher Name                +
               EXITNME=ENFLST01,  -- Exit Name                       +
               DTOKEN=ENFL01T     -- Returned Token Field
*
* Check the return code from ENFREQ - if not zero issue message
*
* - Local variables
DATAAREA DSECT
ENFL01@  DS    A
ENFL01T  DS    F
ENFPTR   DS    A
LPARM    DS    CL16
*
* - Local constants
ENFLSTEN CSECT
         DS    0F
ENFC35   EQU   35
THISMOD  DC    CL8'ENFLSTEN'
ENFLST01 DC    CL8'ENFLST01'
*
* - External control blocks
         CVT   DSECT=YES
         IEFENFCT
ENFLSTEN CSECT
         IXCYENF
DATAAREA DSECT
         IEFENFPM
LENODATA EQU   *-DATAAREA
*
```

DELETE option

Syntax

The standard form of the ENFREQ macro for ACTION=DELETE is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENFREQ.
ENFREQ	
ƀ	One or more blanks must follow ENFREQ.

Syntax	Description
ACTION=DELETE	
,CODE=*event code*	*event code*: Decimal digit.
,DTOKEN=*dtoken*	*dtoken*: RX-type address or address in register (2) - (12).
,RELATED=*(value)*	*value*: Any text.

Parameters

The parameters are explained as follows:

ACTION=DELETE
A required parameter that specifies that you want to delete an existing request to listen for a specified event. When a program issues ENFREQ with the ACTION=DELETE parameter, ENF either deletes the listen request immediately if the listener user exit has completed, or waits until the listener user exits completes. Because the listener user exit might not have completed processing at the time the delete request is issued, do not release the listener user exit's storage.

,CODE=*event code*
A required parameter that specifies the ENF event for which a program no longer needs notification. The *event code* can be any of the decimal codes listed in Table 1.

,DTOKEN=*dtoken*
The required parameter that identifies the specific listen request you are deleting. The system returned the token when you issued the ACTION=LISTEN request.

,RELATED=*(value)*
An optional parameter that specifies information used to self-document macros by 'relating' functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and can be any valid coding values.

Return and reason codes

For the return codes, in hexadecimal, from the ENFREQ macro see "Return codes" on page 59.

On systems running z/OS V2R1 or higher, for return code of 0 or 28 (X'1C') from a ACTION=DELETE request, a reason code is provided in access register 15:

0
The ACTION=DELETE request has completed. The listen exit is not executing and will not be called again.

1
The ACTION=DELETE request is pending. The listen exit may be executing or may be called again.

The reason code provides a way to determine when it is safe to free or reuse storage containing the exit or used by the exit. Storage can be safely freed or reused when the first ACTION=DELETE request provides return code 0 and reason code 0, or after the first ACTION=DELETE request provides return code 0 and reason code 1, a subsequent ACTION=DELETE request provides return code 28 (X'1C) and reason code 0.

Because there is no way to determine when it is safe to free or reuse storage containing the exit or used by the exit on systems running z/OS V1R13 or earlier, such storage should never be freed or reused.

ENFREQ ACTION=LISTEN - List form

Use the list form of the ENFREQ macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro constructs a parameter list that the execute form of the macro can use or modify.

Syntax

The list form of the ENFREQ macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENFREQ.
ENFREQ	
ƀ	One or more blanks must follow ENFREQ.
ACTION=LISTEN	
,CODE=*event code*	*event code*: Decimal digit.
,MASEXIT=No	**Default:** MASEXIT=NO
,MASEXIT=YES	
,FLTRBLK=*filter block addr*	*filter block addr*: the address of the filter block
,QUAL=*qualifier*	*qualifier*: A constant value
,QMASK=*qmask keywords*	*qmask keywords*: BYTE1, BYTE2, BYTE3, BYTE4, ALL, NONE.
	Default: QMASK=NONE
,BITQUAL=*bitqual*	*bitqual*: name of a 32-byte field, hexadecimal numeric value (X'xxx'),
,BITCOMPARE=SUBSET	**Default:** BITCOMPARE=SUBSET
,BITCOMPARE=INTERSECT	
,BITCOMPARE=EQUAL	

Syntax	Description
,SRBEXIT=*exitrtn addr*	*exitrtn addr*: A-type address.
,EXIT=*exitrtn addr*	*exitrtn addr*: A-type address.
,PARM=*parm addr*	*parm addr*: A-type address.
,PARM=*parm data*	*parm data*: a fullword of data
,EOT=*NO*	**Default:** EOT=NO.
,EOT=*YES*	
,EOM=*NO*	**Default:** EOM=NO.
,EOM=*YES*	
,PLISTVER=2	**Default:** Version implied by keywords
,PLISTVER=3	
,PLISTVER=MAX	
,RELATED=*(value)*	*value*: Is any text.
,XSYS=*NO*	**Default:** XSYS=NO.
,XSYS=*YES*	
,MF=L	

Parameters

The parameters are explained under the standard form of the ENFREQ macro with ACTION=LISTEN, with the following exceptions:

,MF=L
Specifies the list form of the ENFREQ macro with ACTION=LISTEN.

ENFREQ ACTION=LISTEN - Execute form

Use the execute form of the ENFREQ macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro uses or modifies the parameter list that the list form built.

Syntax

The execute form of the ENFREQ macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENFREQ.
ENFREQ	
ƀ	One or more blanks must follow ENFREQ.
ACTION=LISTEN	
,CODE=*event code*	*event code*: Decimal digit.
,DTOKEN=*dtoken addr*	*dtoken addr*: RX-type address or address in register (2) - (12).
,ESTBNME=*estab name*	*estab name*: RX-type address or address in register (2) - (12).
,EXITNME=*exitrtn name*	*exitrtn name:* RX-type address or address in register (2) - (12).
,FLTRBLK=*filter block addr*	*filter block addr:* RX-type address or address in register (2) - (12).
,QUAL=*qualifier*	*qualifier*: A four-byte value.
,QMASK=*qmask keywords*	*qmask keywords*: BYTE1, BYTE2, BYTE3, BYTE4, ALL, NONE.
	Default: QMASK=NONE
,BITQUAL=*bitqual*	*bitqual*: name of a 32-byte field, hexadecimal numeric value (X'xxx'), or address in register (2) - (12).
,BITCOMPARE=SUBSET	**Default:** BITCOMPARE=SUBSET
,BITCOMPARE=INTERSECT	
,BITCOMPARE=EQUAL	
,SRBEXIT=*exitrtn addr*	*exitrtn addr*: A-type address, or address in register (2) - (12).
,EXIT=*exitrtn addr*	*exitrtn addr*: A-type address or address in register (2) - (12).

Syntax	Description
,PARM=*parm addr*	*parm addr*: A-type address, or address in register (2) - (12).
,PARM=*parm data*	*parm data*: a fullword of data
,EOT=*NO*	**Default:** EOT=NO.
,EOT=*YES*	
,EOM=*NO*	**Default:** EOM=YES.
,EOM=*NO*	
,PLISTVER=2	**Default:** Version implied by keywords
,PLISTVER=3	
,PLISTVER=MAX	
,RELATED=*(value)*	*value*: Is any text.
,XSYS=*NO*	**Default:** XSYS=NO.
,XSYS=*YES*	
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).

Parameters

The parameters are explained under the standard form of the ENFREQ macro with ACTION=LISTEN, with the following exceptions:

,MF=(E,*list addr*)
Specifies the execute form of the ENFREQ macro with ACTION=LISTEN.

list addr specifies the area that the system uses to store the parameters.

ENFREQ ACTION=DELETE - List form

Use the list form of the ENFREQ macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro constructs a parameter list that the execute form of the macro can use or modify.

Syntax

The list form of the ENFREQ macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.

Syntax	Description
ƀ	One or more blanks must precede ENFREQ.
ENFREQ	
ƀ	One or more blanks must follow ENFREQ.
ACTION=DELETE	
,CODE=*event code*	*event code*: Decimal digit.
,RELATED=*(value)*	*value*: Any text.
,MF=L	

Parameters

The parameters are explained under the standard form of the ENFREQ macro with ACTION=DELETE, with the following exceptions:

,MF=L
Specifies the list form of the ENFREQ macro with ACTION=DELETE.

ENFREQ ACTION=DELETE - Execute form

Use the execute form of the ENFREQ macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro uses or modifies the parameter list that the list form built.

Syntax

The execute form of the ENFREQ macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENFREQ.
ENFREQ	
ƀ	One or more blanks must follow ENFREQ.

Syntax	Description
ACTION=DELETE	
,CODE=*event code*	*event code*: Decimal digit.
,DTOKEN=*dtoken addr*	*dtoken addr*: RX-type address or address in register (2) - (12).
,RELATED=*(value)*	*value*: Any text.
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).

Parameters

The parameters are explained under the standard form of the ENFREQ macro with ACTION=DELETE, with the following exceptions:

,MF=(E,*list addr*)
Specifies the execute form of the ENFREQ macro with ACTION=DELETE.

list addr specifies the area that the system uses to contain the parameters.

Chapter 4. ENQ — Request control of a serially reusable resource

Description

ENQ assigns control of one or more serially reusable resources to a task. If any of the resources are not available, the task might be placed in a wait condition until all of the requested resources are available. Once control of a resource has been assigned to a task, it remains with that task until one of the programs running under that task issues a DEQ macro to release the resource or the task terminates.

You can request either shared or exclusive use of a resource. ENQ identifies the resource by a pair of names, the *qname* and the *rname*, and a scope value. The scope value determines what other tasks, address spaces, or systems can use the resource. All programs that share the resource must use the qname, rname, and scope value consistently.

Use ENQ with RET=TEST to determine the status of the resource. Return codes tell whether the resource is immediately available or in use, and whether control has been previously requested by the active task in another ENQ macro.

ENQ with the MASID and MTCB parameters allows a further conditional control of a resource. One task, called the "issuing task" can issue an ENQ macro for a resource specifying the ASID and TCB of another task, called the "matching task". MTCB and MASID parameters are specified with RET=HAVE, RET=TEST, or ECB to provide additional return codes. If the issuing task does not receive control of the resource, it may receive a return code indicating that the resource is controlled by the matching task. Upon receiving this return code, the issuing task could use the resource, if serialization between itself and the matching task has been prearranged through a protocol.

Global resource serialization counts and limits the number of concurrent resource requests from an address space. If an unconditional ENQ (an ENQ that uses the RET=NONE option) causes the count of concurrent resource requests to exceed the limit, the caller ends abnormally with a system code of X'538'. For more information, see Limiting concurrent requests for resources in *z/OS MVS Programming: Assembler Services Guide*.

Unless you specify otherwise, when a global resource serialization complex is initialized, global resource serialization searches the SYSTEM inclusion resource name list (RNL) and the SYSTEMS exclusion RNL for every resource specified with a scope of SYSTEM or SYSTEMS. A resource whose name appears on one of these RNLs might have its scope changed from the scope that appears on the macro. To prevent RNL processing, use the RNL=NO parameter. See *z/OS MVS Planning: Global Resource Serialization* for additional information about RNL processing.

The ENQ macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the SMC, ECB, TCB, MASID, and MTCB parameters. For information about using the ENQ macro to serialize resources, see Serialization in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for callers of ENQ are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key. For the SMC, ECB, TCB, MASID, and MTCB parameters or when the specified *qname* is ADRDFRAG, ADRDSN, ARCENQG, BWODSN, SYSZ*, SYSCTLG, SYSDSN, SYSIEA01, SYSIEECT, SYSIEFSD, SYSIGGV1, SYSIGGV2, SYSPSWRD, SYSVSAM, or SYSVTOC, the authorization must be **one of the following**: • Supervisor state • PSW key 0-7 • APF-authorized.
Dispatchable unit mode:	Task
Cross memory mode:	For LINKAGE=SVC: PASN=HASN=SASN For LINKAGE=SYSTEM: Any PASN, Any HASN, Any SASN For LINKAGE=SYSTEM with SMC=STEP: PASN=HASN, Any SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space. Except for the TCB, all parameters can reside above 16 megabytes.

Programming requirements

None.

Restrictions

See Avoiding interlock in *z/OS MVS Programming: Assembler Services Guide* to ensure that you are following the protocols required to prevent the interlock.

Issuing two ENQ macros for the same resource without an intervening DEQ macro causes the task to end abnormally, unless the second ENQ designates RET=TEST, USE, CHNG, or HAVE. If the task ends, either normally or abnormally, while the task still has control of any serially reusable resources, all requests made by this task automatically have DEQ processing performed for them. If resource input addresses are incorrect, the task abnormally ends.

The caller cannot have an EUT FRR established.

There are some considerations to be aware of when using enclaves for tasks that serialize resources using the ENQ macro. For details, see Using ENQ/DEQ or latch manager services with enclaves in *z/OS MVS Programming: Workload Management Services*.

Input register information

Before issuing the ENQ macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
One of the following:

- If you specify RET=TEST, RET=USE, RET=CHNG, RET=HAVE, or ECB: If all return codes for the resources named in the ENQ macro are 0, register 15 contains 0. If any of the return codes are not 0, register 15 contains the address of a storage area containing the return codes.
- Otherwise: Used as a work register by the system.

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the ENQ macro is described as follows.

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENQ.
ENQ	
ƀ	One or more blanks must follow ENQ.
(	
qname addr	*qname addr*: A-type address or register (2) - (12).

Syntax	Description
,	
,*rname addr*	*rname addr*: A-type address or register (2) - (12).
,	**Default**: E
,E	
,S	
,	
,*rname length*	*rname length*: symbol, decimal digit, or register (2) - (12). **Default**: assembled length of *rname* **Note**: Code *rname length* if *rname addr* is a register.
,	
,STEP	**Default**: STEP
,SYSTEM	
,SYSTEMS	
)	
,RET=CHNG	**Default**: RET=NONE
,RET=HAVE	
,RET=TEST	
,RET=USE	
,RET=NONE	
,SMC=NONE	**Default**: SMC=NONE
,SMC=STEP	
,ECB=*ecb addr*	*ecb addr*: A-type address or register (2) - (12).
,TCB=*tcb addr*	*tcb addr*: A-type address or register (2) - (12).
	Note: Do not specify ECB with RET. You can specify ECB and TCB together. If TCB is specified without ECB, you must specify RET=CHNG, TEST or USE.
,MASID=*matching-asid addr*	*matching-asid addr*: A-type address or register (2) - (12).
	Note: MTCB is required with MASID. Do not specify SMC or TCB with MASID.

Syntax	Description
,MTCB=*matching-tcb addr*	*matching-tcb addr*: A-type address or register (2) - (12).
	Note: MASID is required with MTCB.
,RNL=YES	**Default**: RNL=YES
,RNL=NO	
,RELATED=*value*	*value*: any valid macro keyword specification.
,LINKAGE=SVC	**DEFAULT**: LINKAGE=SVC
,LINKAGE=SYSTEM	

Parameters

The parameters are explained as follows:

(
Specifies the beginning of the resource description.

qname addr
Specifies the address of an 8-character name. The name can contain any valid hexadecimal character. Every program issuing a request for a serially reusable resource must use the same *qname, rname*, and scope to represent the resource. Some names, such as those beginning with certain letter combinations (SYSZ for example), are used to protect system resources by requiring that the issuing program be in supervisor state, or system key, or APF-authorized. Authorized programs should use a restricted qname (as described under Minimum authorization in the Environment topic of this chapter) to prevent interference from unauthorized programs.

Note: See *z/OS MVS Diagnosis: Reference* for a list of major and minor ENQ/DEQ names and the resources that issue the ENQ/DEQ.

,
,rname addr
Specifies the address of the name used together with *qname* to represent a single resource. The name must be 1 - 255 bytes long. Each byte can have any value from X'00' to X'FF'.

,
,E
,S
Specifies whether the request is for exclusive (E) or shared (S) control of the resource. If the resource is modified while under control of the task, the request must be for exclusive control; if the resource is not modified, the request should be for shared control.

,
,rname length
Specifies the length of the *rname*. If this parameter is omitted, the system uses the assembled length of the *rname*. To override the assembled length, specify this parameter.

The value you can code depends on whether you also specify MASID and MTCB:

- If you specify MASID and MTCB, you can code a value 1 - 128.
- If you do not specify MASID and MTCB, you can code a value 1 - 255.

In either case, you can specify 0, which means that the length of the *rname* must be contained in the first byte at the *rname addr*.

,
,STEP
,SYSTEM
,SYSTEMS
Specifies the scope of the resource.

STEP specifies that the resource can be used only within an address space. If STEP is specified, a request for the same *qname* and *rname* from a program in another address space denotes a different resource.

SYSTEM specifies that the resource can be used by programs in more than one address space.

SYSTEMS specifies that the resource can be shared between systems.

STEP, SYSTEM, and SYSTEMS are mutually exclusive and do not refer to the same resource. If two macros specify the same *qname* and *rname*, but one specifies STEP and the other specifies SYSTEM or SYSTEMS, they are treated as requests for different resources.

)
Specifies the end of the resource description.

Notes on specifying multiple resources on one ENQ request:

- Within a single set of parentheses, you can repeat the *qname addr*, *rname addr*, type of control, *rname length*, and the scope until there is a maximum of 255 characters, including the parentheses.
- The following parameters apply to all the resources you specify on the request: RET, SMC, ECB, TCB, MASID, MTCB, and RNL.

,RET=CHNG
,RET=HAVE
,RET=TEST
,RET=USE
,RET=NONE
Specifies the type of request for the resources named on the ENQ request.

CHNG
The status of the resource specified is changed from shared to exclusive control. When RET=CHNG is specified, the exclusive|shared (E|S) parameter is overidden. This parameter ensures that the request will be exclusive regardless of the other parameter.

HAVE
Control of the resources is requested conditionally; that is, control is requested only if a request has not been made previously for the same task.

TEST
The availability of the resources is to be tested, but control of the resources is not requested.

USE
Control of the resources is to be assigned to the active task only if the resources are immediately available. If any of the resources are not available, the active task is not placed in a wait condition. When SYNCRES is enabled, the request is subject to a delay for the reserve even if RET=USE or ContentionAct=Fail is specified.

NONE
Control of all the resources is unconditionally requested.

See Return and reason codes for an explanation of the return codes for these requests.

,SMC=NONE
,SMC=STEP
,ECB=*ecb addr*
,TCB=*tcb addr*
Specifies optional parameters available to the system programmer:

SMC specifies that the set must-complete function is not to be used (NONE) or that it is to set as non-dispatchable other tasks for the step until the requesting task has completed its operations on the resource (STEP).

See Using the must-complete function (ENQ/DEQ) in *z/OS MVS Programming: Authorized Assembler Services Guide* for a description of the set must-complete function.

Do not use SMC or RET with ECB.

When SMC=STEP is specified with RET=HAVE and the requesting task already has control of the resource, the SMC function is turned on and the task continues to control the resource.

SMC and TCB are mutually exclusive with the MASID parameter; therefore, hexadecimal return codes 20, 24, 28, and 44 will not be given by an ENQ using the SMC or TCB operands.

The return codes and status of the set must-complete function for the various RET specifications are as follows:

RET Parameter	Hexadecimal Code	SMC Status
RET=CHNG	0	on
RET=CHNG	4	off
RET=CHNG	8	off
RET=CHNG	14	off
RET=HAVE	0	on
RET=HAVE	8	on
RET=HAVE	14	off
RET=HAVE	18	off
RET=TEST	0	off
RET=TEST	4	off
RET=TEST	8	off
RET=TEST	14	off
RET=USE	0	on
RET=USE	4	off
RET=USE	8	off
RET=USE	14	off
RET=USE	18	off

ECB specifies the address of an ECB, and conditionally requests all of the resources named in the macro. If the return code for one or more requested resources is hexadecimal 4 or 24 and the request is not nullified by a corresponding DEQ, the ECB is posted when all the requested resources (specifically, those that initially received a return code of 4 or 24) are assigned to the requesting task.

If the ECB parameter is an A-type address, the address is the name of the fullword that is used as an ECB. If the operand is a register, then the register contains the address of the ECB.

Note: The ECB must reside in storage that is addressable from the caller's home address space.

TCB specifies a register that points to a TCB or specifies the address of a fullword on a fullword boundary that points to a TCB on whose behalf the ENQ is to be done. If TCB is specified, one of the following must also be specified:

- RET=TEST

- RET=USE
- RET=CHNG
- ECB

Note: The TCB resides in storage below 16 megabytes in the caller's address space.

,MASID=*matching-asid addr*
Specifies the matching task (by defining a matching ASID) for the ENQ, if it is used together with the MTCB parameter. MASID defines the ASID of a task that may be using a resource desired by the caller. If the MASID parameter is an A-type address, the address is the name of a fullword containing the ASID. If the operand is a register, then the register contains the ASID.

,MTCB=*matching-tcb addr*
Specifies the matching task (by defining a matching TCB) for the ENQ, if used together with the MASID parameter. MTCB defines the TCB of a task that may be using a resource desired by the caller. If the MTCB parameter is an A-type address, the address is the name of a fullword containing the TCB. If the operand is a register, then the register contains the TCB.

If the task specified by the MASID and MTCB parameters is not using the resource, global resource serialization gives control to the caller and returns a return code indicating whether the resource can be used. If the task specified by MASID and MTCB parameters is using the resource, global resource serialization records a request for the resource, suspends the issuing task until the resource is available, or optionally returns a return code indicating that an ECB will be posted when the resource can be used.

The MASID and MTCB parameters are specified with RET=HAVE, RET=TEST, or ECB parameters to elicit additional return codes that provide information about the owner of the resource.

See the description of the *rname length* for information about specifying *rname length* with MASID and MTCB.

,RNL=YES
,RNL=NO
Controls global resource serialization RNL processing, which can cause the scope value of a resource to change. In general, IBM recommends that you use the default, RNL=YES, to allow global resource serialization to perform RNL processing. Use RNL=NO when you are sure that you want the request to be processed only by global resource serialization using only the specified scope or when an alternate serialization product or installation exit should be prevented from altering the scope, extending the scope beyond the GRS complex, or restricting the scope to systems other than all of those in the GRS complex. See RNL processing in *z/OS MVS Planning: Global Resource Serialization* for more information about the use of RNL=NO.

,RELATED=*value*
Specifies information used to self-document macros by ‘relating’ functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

,LINKAGE=SVC
,LINKAGE=SYSTEM
Specifies the type of linkage the caller is using to invoke the ENQ service.

For LINKAGE=SVC, the linkage is through an SVC instruction. This linkage is valid only when the caller is in primary mode and the primary, home, and secondary address spaces are the same.

For LINKAGE=SYSTEM, the linkage uses a non-SVC entry. This linkage is valid in cross memory mode or in non-cross memory mode. LINKAGE=SYSTEM is intended to be used by programs in cross memory mode.

- If ECB= is specified, the ECB (not the address of the ECB) must be addressable from the home address space.
- If TCB= is specified, then the specified TCB in the home address space is associated with the resource; otherwise, the TCB in the home address space making the request is associated with the resource.

The default is LINKAGE=SVC.

ABEND codes

For only unconditional requests, the caller might encounter abend code X'138' or X'538'. For unconditional or conditional requests, the caller might encounter one of the following abend codes:

- X'238'
- X'338'
- X'438'
- X'738'
- X'838'
- X'938'

See *z/OS MVS System Codes* for explanations and responses for these codes.

Return and reason codes

The system provides a return code only if you specify RET=TEST, RET=USE, RET=CHNG, RET=HAVE, or ECB; otherwise, return of the task to the active condition indicates that control of the resource has been assigned or was previously assigned to the task. If all return codes for the resources named in the ENQ macro are 0, register 15 contains 0. For nonzero return codes, register 15 contains the address of a storage area containing the return codes, as shown in Figure 1.

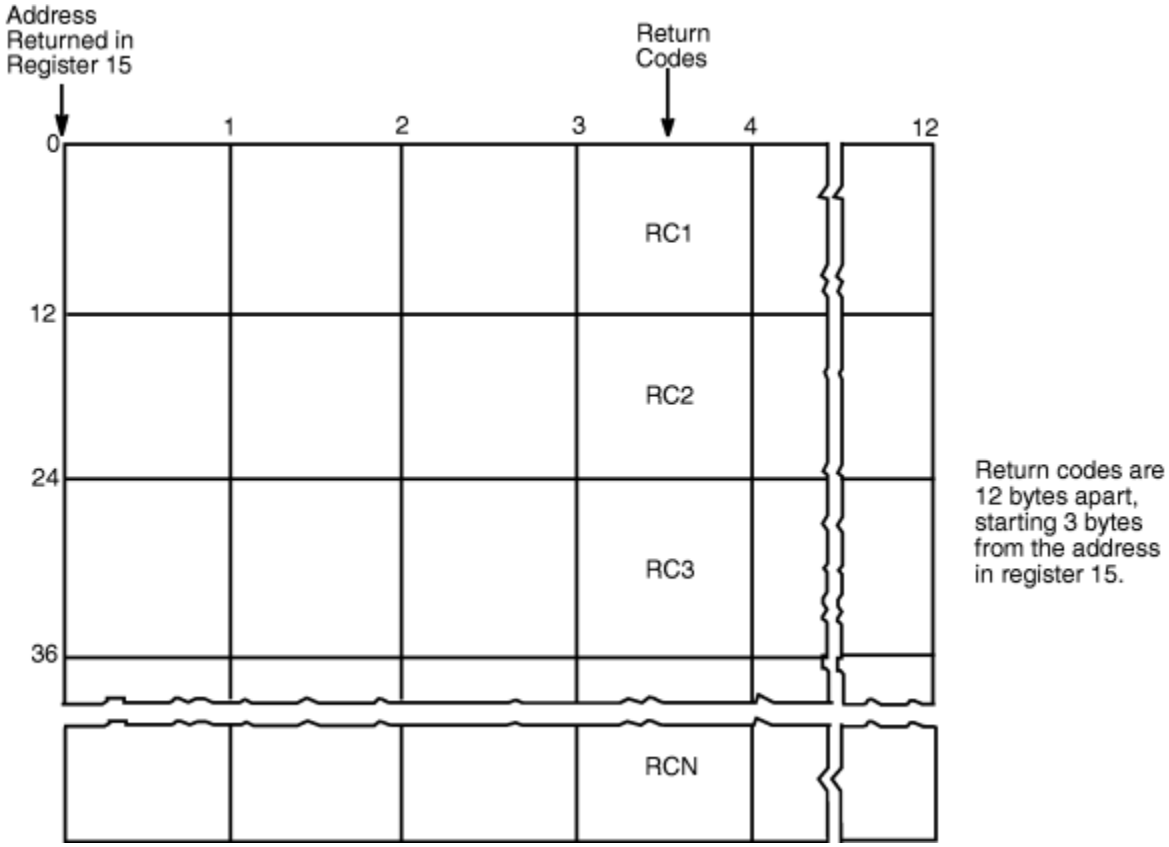


Figure 4. Return Code Area Used by ENQ

The return codes are placed in the parameter list resulting from the macro expansion in the same sequence as the resource names in the ENQ macro.

The return codes for the ENQ macro with the RET=TEST parameter are described in Table 1.

Table 6. Return Codes for the ENQ Macro with the RET=TEST Parameter	
Hexadecimal Return Code	**Meaning and Action**
0	**Meaning**: The resource is immediately available. **Action**: None required. However, you might take some action based on your application.
4	**Meaning**: The resource is not immediately available. **Action**: None required. However, you might take some action based on your application.
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: None required. However, you might take some action based on your application. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.
20	**Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource. **Action**: None required. However, you might take some action based on your application.

The return codes for the ENQ macro with the RET=USE parameter are described in Table 2.

Table 7. Return Codes for the ENQ Macro with the RET=USE Parameter	
Hexadecimal Return Code	**Meaning and Action**
0	**Meaning**: The active task now has control of the resource. **Action**: None.
4	**Meaning**: The resource is not immediately available. **Action**: None required. However, you might take some action based on your application.
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: None required. However, you might take some action based on your application. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.
18	**Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. **Action**: Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.

The return codes for the ENQ macro with the RET=CHNG parameter are described in Table 3.

Table 8. Return Codes for the ENQ Macro with the RET=CHNG Parameter	
Hexadecimal Return Code	**Meaning and Action**
0	**Meaning**: The status of the resource has been changed to exclusive. **Action**: None.

Table 8. Return Codes for the ENQ Macro with the RET=CHNG Parameter (continued)	
Hexadecimal Return Code	**Meaning and Action**
4	**Meaning**: The status of the resource cannot be changed to exclusive. Other tasks share the resource. **Action**: None required. However, you might take some action based on your application.
8	**Meaning**: The status of the resource cannot be changed to exclusive. Either no tasks have issued an ENQ request for the resource, or the task acquired the resource through the MASID parameter. **Action**: None required. However, you might take some action based on your application.
14	**Meaning**: The status of the resource cannot be changed to exclusive. A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.

The return codes for the ENQ macro with the RET=HAVE parameter are described in Table 4.

Table 9. Return Codes for the ENQ Macro with the RET=HAVE Parameter	
Hexadecimal Return Code	**Meaning and Action**
0	**Meaning**: The active task now has control of the resource. **Action**: None.
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: None required. However, you might take some action based on your application. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task but that request has not yet been satisfied (such as an ENQ with RET=NONE which waits for the resource). The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.
18	**Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. **Action**: Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.
20	**Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource. **Action**: The caller can use the resource, but it must ensure that the owning task does not terminate while the caller is using the resource. If the caller requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the caller requested shared control and the owning task requested shared control, this return code might indicate that a previous task had requested exclusive control. The caller must issue a DEQ macro to cancel this ENQ request.
28	**Meaning**: The caller cannot obtain exclusive control of the resource using the ENQ macro with the MASID and MTCB parameters. The matching task's involvement with other tasks precludes control by the caller. **Action**: This task must not issue a DEQ macro to cancel the ENQ request.

Table 9. Return Codes for the ENQ Macro with the RET=HAVE Parameter (continued)	
Hexadecimal Return Code	**Meaning and Action**
44	**Meaning**: The caller is violating a restriction of using the ENQ macro with the MASID and MTCB parameters in one or more of the following ways: • Another task has already issued the ENQ macro for this resource specifying the same values for the MASID and MTCB parameters • The MASID and MTCB parameters specify a task that acquired control of the resource by using the ENQ macro with the MASID and MTCB parameters • The matching task requested ownership of the resource but has not yet been granted ownership. **Action**: Do not use the resource; the caller does not have control of it.

The return codes for the ENQ macro with the ECB parameter are described in Table 5.

Table 10. Return Codes for the ENQ Macro with the ECB Parameter	
Hexadecimal Return Code	**Meaning and Action**
0	**Meaning**: The active task now has control of the resource. **Action**: Do not wait on the ECB; it will not be posted.
4	**Meaning**: The active task does not have control of the resource yet. The ECB will be posted when the system assigns control to that task. **Action**: Wait on the ECB if your program cannot continue processing without control of the resource.
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: Do not wait on the ECB; it will not be posted. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: Do not wait on the ECB; it will not be posted.
18	**Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. **Action**: Do not wait on the ECB; it will not be posted. Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.
20	**Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource. **Action**: Do not wait on the ECB; it will not be posted. The caller can use the resource, but it must ensure that the owning task does not terminate while the caller is using the resource. If the caller requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the caller requested shared control and the owning task requested shared control, this return code might indicate that a previous task had requested exclusive control. The caller must issue a DEQ macro to cancel this ENQ request.
24	**Meaning**: The caller that specifies the ENQ macro with the MASID and MTCB parameters will have exclusive control after the ECB is posted. **Action**: Wait on the ECB. Once the ECB is posted, the caller may use the resource, but must ensure that the matching task does not terminate while the caller is using the resource. The caller must issue a DEQ macro to cancel the ENQ request.
28	**Meaning**: The caller cannot obtain exclusive control of the resource using the ENQ macro with the MASID and MTCB parameters. The matching task's involvement with other tasks precludes control by the caller. **Action**: Do not wait on the ECB; it will not be posted. The caller must not issue a DEQ macro to cancel the ENQ request.

Table 10. Return Codes for the ENQ Macro with the ECB Parameter (continued)	
Hexadecimal Return Code	**Meaning and Action**
44	**Meaning**: The caller is violating a restriction of using the ENQ macro with the MASID and MTCB parameters in one or more of the following ways: • Another task has already issued the ENQ macro for this resource specifying the same values for the MASID and MTCB parameters • The MASID and MTCB parameters specify a task that acquired control of the resource by using the ENQ macro with the MASID and MTCB parameters • The matching task requested ownership of the resource but has not yet been granted ownership. **Action**: Do not wait on the ECB; it will not be posted. Do not use the resource; the caller does not have control of it.

Example 1

Unconditionally request exclusive control of a serially reusable resource that is known only within the address space (STEP), and set to non-dispatchable other tasks for the step until the requesting task has completed its operations on the resource.

```
ENQ  (MAJOR1,MINOR1,E,8,STEP),SMC=STEP
```

Example 2

Conditionally request control of a resource that can be shared on behalf of another task. The resource is known by more than one address space, and is only wanted if immediately available.

```
ENQ  (MAJOR2,MINOR2,S,4,SYSTEM),TCB=(R2),RET=USE
```

ENQ - List form

Use the list form of ENQ to construct a control program parameter list. You can specify any number of resources on ENQ, therefore, the number of *qname, rname*, and scope combinations in the list form of the ENQ macro must be equal to the maximum number of *qname, rname*, and scope combinations in any execute form of the macro that refers to that list form.

Syntax

The list form of the ENQ macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENQ.
ENQ	
ƀ	One or more blanks must follow ENQ.
(	

Syntax	Description
qname addr	*qname addr*: A-type address or register (2) - (12).
,	
,rname addr	*rname addr*: A-type address or register (2) - (12).
,	**Default**: E
,E	
,S	
,	
,rname length	*rname length*: symbol or decimal digit.
	Default: assembled length of *rname*
,	**Default**: STEP
,STEP	
,SYSTEM	
,SYSTEMS	
)	
,RET=CHNG	**Default**: RET=NONE
,RET=HAVE	
,RET=TEST	
,RET=USE	
,RET=NONE	
,SMC=NONE	**Default**: SMC=NONE
,SMC=STEP	
,ECB=*ecb addr*	*ecb addr*: A-type address.
,TCB=0	**Note**: ECB cannot be specified with RET. **Note**: TCB or ECB must be specified on the list form if it is used on the execute form. ECB and TCB can be specified together. If you specify TCB without ECB, specify RET=CHNG, TEST or USE.
,MASID=*0*	**Note**: MTCB is required with MASID. Do not specify SMC or TCB with MASID.
,MTCB=*0*	**Note**: MASID is required with MTCB.

Syntax	Description
,RNL=YES	**Default**: RNL=YES
,RNL=NO	
,RELATED=*value*	*value*: any valid macro keyword specification.
,MF=L	

Parameters

The parameters are explained under the standard form of the ENQ macro, with the following exception:

,MF=L
Specifies the list form of the ENQ macro.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.

ENQ - Execute form

A remote control program parameter list is used in and can be modified by the execute form of the ENQ macro. The parameter list must be generated by the list form of ENQ.

Syntax

The execute form of the ENQ macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ENQ.
ENQ	
ƀ	One or more blanks must follow ENQ.
(	**Note**: (and) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired then (,), and all parameters between (and) should not be specified. If something in the list is desired, the (,), and all parameters in the list should be specified as indicated at the left.
qname addr	*qname addr*: RX-type address or register (2) - (12).
,	

Syntax	Description
,*rname addr*	*rname addr*: RX-type address or register (2) - (12).
,	
,E	
,S	
,	
,*rname length*	*rname length*: symbol, decimal digit, or register (2) - (12).
,	
,STEP	
,SYSTEM	
,SYSTEMS	
)	**Note**: See note opposite (above.
,RET=CHNG	
,RET=HAVE	
,RET=TEST	
,RET=USE	
,RET=NONE	
,SMC=NONE	*ecb addr*: RX-type address or register (2) - (12).
,SMC=STEP	*tcb addr*: RX-type address or register (2) - (12).
,ECB=*ecb addr*	**Note**: ECB cannot be specified with RET above.
,TCB=*tcb addr*	**Note**: ECB and TCB can be specified together. If you specify TCB without ECB, then specify RET=CHNG, TEST, or USE.
,MASID=*matching-asid addr*	*matching-asid addr*: RX-type address or register (2)-(12).
	Note: MTCB is required with MASID. Do not specify SMC or TCB with MASID.
,MTCB=*matching-tcb addr*	*matching-tcb addr*: RX-type address or register (2)-(12).
	Note: MASID is required with MTCB.
,RNL=YES	
,RNL=NO	
,RELATED=*value*	*value*: any valid macro keyword specification.

Syntax	Description
,LINKAGE=SVC	**DEFAULT**: LINKAGE=SVC
,LINKAGE=SYSTEM	
,MF=(E,*list addr*)	*list addr*: RX-type address or register (1) - (12).

Parameters

The parameters are explained under the standard form of the ENQ macro, with the following exceptions:

,MF=(E,*list addr)*
Specifies the execute form of the ENQ macro.

list addr specifies the area that the system uses to contain the parameters.

Note: If ECB (or TCB) is specified in the execute form, ECB (or TCB=0) must be specified in the list form. If MASID and MTCB are specified, MASID=0 and MTCB=0 must be specified in the list form.

The list form of this macro generates a prefix followed by the parameter list, however the label specified in MF=L does not include an offset prefix area. If MASID, MTCB, TCB, or ECB is specified, these labels are offset; allowance must be made for the parameter list prefix.

Chapter 5. ESPIE — Extended SPIE

Description

The ESPIE macro extends the function of the SPIE (specify program interruption exits) macro to callers in 31-bit and 64-bit addressing mode. For additional information concerning the relationship between the SPIE and the ESPIE macros, see the information on program interruptions in *z/OS MVS Programming: Assembler Services Guide* and *z/OS MVS Programming: Authorized Assembler Services Guide*.

The ESPIE macro performs the following functions using the options specified:

- Establishes an ESPIE environment (that is, identifies the interruption types that are to cause entry to the ESPIE exit routine) by executing the SET option of the ESPIE macro
- Deletes an ESPIE environment (that is, cancels the current SPIE/ESPIE environment) by executing the RESET option of the ESPIE macro
- Determines the current SPIE/ESPIE environment by executing the TEST option of the ESPIE macro.

The following description of the ESPIE macro also appears in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

For information about programs in 64-bit addressing mode (AMODE 64), see *z/OS MVS Programming: Extended Addressability Guide*.

The information documented under the following topics is provided separately for each of the three options (SET, RESET, and TEST):

- "Input Register Information"
- "Output Register Information"
- "Syntax"
- "Parameters"
- "Return and Reason Codes"
- "Examples"

The information documented in the following topics applies to all three options of the ESPIE macro (SET, RESET, and TEST):

- "Environment"
- "Programming Requirements"
- "Restrictions"
- "Performance Implications"
- "ABEND Codes"

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	To issue ESPIE without encountering an abnormal end, callers must be in problem state, with a PSW key value that is equal to the TCB assigned key, except when ESPIE RESET is issued or ESPIE SET is issued with no interruption codes specified (where key 0 supervisor state is allowed).
Dispatchable unit mode:	Task

Environmental factor	Requirement
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Performance implications

Programs that need to intercept only specific hardware program check interruptions (such as arithmetic exceptions or data conversion exceptions) will find ESPIE to be more efficient than establishing an ESTAE environment to screen all abends for specific OCx abends.

ABEND codes

ESPIE might return with abend code X'46D'. See *z/OS MVS System Codes* for an explanation and programmer responses.

ESPIE SET option

Input register information

Before issuing the SET option of the ESPIE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain the following information:

Register
Contents

0
Used as a work register by the system

1
Token representing the previously active SPIE/ESPIE environment

2-13
Unchanged

14
Used as a work register by the system

15
Return code of 0

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Syntax

The standard form of the ESPIE macro with the SET option is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ESPIE.
ESPIE	
␣	One or more blanks must follow ESPIE.
SET	
,exit addr	*exit addr*: A-type address or register (2) - (12).
,(interruptions)	*interruptions*: Decimal numbers 1 - 15, expressed as: • single values: (2, 3, 4, 7, 8, 9, 10) • ranges of values: ((2, 4), (7, 10)) • combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr*: A-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

SET
Indicates that an ESPIE environment is to be established.

,exit addr
Specifies the address of the exit routine to be given control when program interruptions of the type specified by *interruptions* occur. The exit routine will receive control in the same addressing mode as the issuer of the ESPIE macro.

,(interruptions)
Indicates the interruption types that are being trapped. The interruption types are:

Number	Interruption Type
1	Operation
2	Privileged operation
3	Execute
4	Protection
5	Addressing
6	Specification
7	Data
8	Fixed-point overflow (maskable)
9	Fixed-point divide
10	Decimal overflow (maskable)
11	Decimal divide
12	Exponent overflow
13	Exponent underflow (maskable)
14	Significance (maskable)
15	Floating-point divide

These interruption types can be designated as one or more single numbers, as one or more pairs of numbers (designating ranges of values), or as any combination of the two forms. For example, (4,8) indicates interruption types 4 and 8; ((4,8)) indicates interruption types 4 through 8.

If a program interruption type is maskable, the corresponding program mask bit in the PSW is set to 1. If a maskable interruption is not specified, the corresponding bit in the PSW is set to 0. Interruption types not specified above are handled by the system. The system forces an abnormal end with the program check as the completion code. If an ESTAE-type recovery routine is also active, the SDWA indicates a system-forced abnormal end. The registers at the time of the error are those of the system.

Note: For ESPIE and SPIE - If you are using vector instructions and an interruption of 8, 12, 13, 14, or 15 occurs, your recovery routine can check the exception extension code (the first byte of the two-byte interruption code in the ESPIE or PIE) to determine whether the exception was a vector or scalar type of exception.

,PARAM=*list addr*
Specifies the fullword address of a parameter list that is to be passed by the caller to the exit routine.

Return and reason codes

None.

Example

Give control to an exit routine for interruption types 1 and 4. EXIT is the location of the exit routine to be given control and PARMLIST is the location of the user-parameter list to be used by the exit routine.

```
ESPIE SET,EXIT,(1,4),PARAM=PARMLIST
```

ESPIE SET - List form

Use the list form of the ESPIE macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters. The list form of ESPIE is valid only for ESPIE SET.

Syntax

The list form of the ESPIE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ESPIE.
ESPIE	
␣	One or more blanks must follow ESPIE.
SET	
,exit addr	*exit addr*: A-type address.
	Note: This parameter must be specified on either the list or the execute form of the macro.
,(interruptions)	*interruptions*: Decimal number 1 - 15, expressed as: • single values: (2, 3, 4, 7, 8, 9, 10) • ranges of values: ((2, 4), (7, 10)) • combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr*: A-type address.

Syntax	Description
,MF=L	

Parameters

The parameters are explained under the standard form of ESPIE SET with the following exception:

,MF=L
Specifies the list form of the ESPIE macro.

Example

Build a nonexecutable problem program parameter list that will cause control to be transferred to the exit routine, EXIT, for the interruption types specified in the execute form of the macro. Provide the address of the user parameter list, PARMLIST.

```
LIST1 ESPIE SET,EXIT,,PARAM=PARMLIST,MF=L
```

ESPIE SET - Execute form

Use the execute form of the ESPIE macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form. The execute form of ESPIE is valid only for ESPIE SET.

Syntax

The execute form of the ESPIE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ESPIE.
ESPIE	
␣	One or more blanks must follow ESPIE.
SET	
,exit addr	*exit addr*: RX-type address or register (2) - (12).
	Note: This parameter must be specified on either the list or the execute form of the macro.

Syntax	Description
,(interruptions)	*interruptions*: Decimal number 1 - 15, expressed as: • single values: (2, 3, 4, 7, 8, 9, 10) • ranges of values: ((2, 4), (7, 10)) • combinations: (2, 3, 4, (7, 10))
,PARAM=*list addr*	*list addr*: RX-type address or register (1) or (2) - (12).
,MF=(E,*ctrl addr)*	*ctrl addr*: RX-type address, or register (1) or (2) - (12).

Parameters

The parameters are explained under the standard form of ESPIE SET with the following exception:

,MF=(E,*ctrl addr*)
Specifies the execute form of the ESPIE macro.

ctrl addr specifies the area that the system uses to store the parameters.

Example

Give control to a installation exit routine for interruption types 1, 4, 6, 7, and 8. The exit routine address and the address of a user parameter list for the exit routine are provided in a remote control program parameter list at LIST1.

```
ESPIE SET,,(1,4,(6,8)),MF=(E,LIST1)
```

ESPIE RESET option

The RESET option of the ESPIE routine cancels the active SPIE/ESPIE environment and restores the SPIE/ESPIE environment specified by *token*.

Input register information

Before issuing the RESET option of the ESPIE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Used as a work register by the system

1
Token identifying the new active SPIE/ESPIE environment

2-13
Unchanged

14
Used as a work register by the system

15
Return code of 0

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Syntax

The RESET option of the ESPIE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESPIE.
ESPIE	
ƀ	One or more blanks must follow ESPIE.
RESET	
,*token*	*token*: RX-type address or register (1) or (2) - (12).

Parameters

The parameters are explained as follows:

RESET
Indicates that the current ESPIE environment is to be deleted and the previously active SPIE/ESPIE environment specified by *token* is to be reestablished.

,token
Specifies a fullword that contains a token representing the previously active SPIE/ESPIE environment. This is the same token that ESPIE processing returned to the caller when the ESPIE trap was established using the SET option of the ESPIE macro.

If the token is zero, all SPIEs and ESPIEs are deleted.

Return and reason codes

None.

Example

Cancel the current SPIE/ESPIE environment and restore the SPIE/ESPIE environment represented by the contents of TOKEN.

```
ESPIE RESET,TOKEN
```

ESPIE TEST option

The TEST option of the ESPIE macro determines the active SPIE/ESPIE environment and returns the information in a 4-byte parameter list.

Input register information

Before issuing the TEST option of the ESPIE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Used as a work register by the system

1-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Syntax

The TEST option of the ESPIE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESPIE.
ESPIE	
ƀ	One or more blanks must follow ESPIE.
TEST	
,parm addr	*parm addr*: RX-type address, or register (1) or (2) - (12).

Parameters

The parameters are explained as follows:

TEST
Indicates a request for information concerning the active or current SPIE/ESPIE environment. ESPIE processing returns this information to the caller in a 4-word parameter list located at *parm addr*.

,parm addr
Specifies the address of a 4-word parameter list aligned on a fullword boundary. The parameter list has the following form:

Word
Content

0
31-bit address of the exit routine (For 24-bit routines, the high order bit is set to 0. For 31-bit routines, the high order bit is set to 1.)

1
Address of the user-defined parameter list

2
Mask of program interruption types

3
Zero

Return and reason codes

ESPIE TEST returns status information about the current ESPIE environment in GPR 15. When control returns from ESPIE TEST, GPR 15 contains one of the following hexadecimal return codes.

Note: These return codes are informational; no actions are required.

Table 11. Return Codes for the ESPIE TEST Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: An ESPIE exit is active and the 4-word parameter list contains the information specified in the description of the *parm addr* parameter.

Table 11. Return Codes for the ESPIE TEST Macro (continued)	
Hexadecimal Return Code	**Meaning**
04	**Meaning**: A SPIE exit is active. Word 1 of the parameter list described under *parm addr* contains the address of the current PICA. Words 0, 2, and 3 of the parameter list contain no relevant information.
08	**Meaning**: No SPIE or ESPIE is active. The contents of the 4-word parameter list contain no relevant information.

Example

Identify the active SPIE/ESPIE environment. Return the information about the exit routine in the 4-word parameter list, PARMLIST. Also return, in register 15, an indication of whether a SPIE, ESPIE, or neither is active.

```
ESPIE TEST,PARMLIST
```

Chapter 6. ESTAE and ESTAEX — Specify task abnormal exit extended

Description

The ESTAE macro provides recovery capability facilities. Issuing the ESTAE macro allows the caller to intercept errors. Control is given to a caller-specified exit routine (called a recovery routine) in which the caller can perform various tasks, including diagnosing the cause of the error and specifying a retry address to avoid abnormal ending.

ESTAE type considerations: The type of ESTAE routine, that is, ESTAE or ESTAEX affects the AMODE of the recovery routine as follows. For recovery routines defined through the:

- ESTAE macro, at the time of entry to the recovery routine, the AMODE will be the same as at the time of invocation of the macro.
- ESTAEX macro, the AMODE will be the same as at the time of invocation of the macro, unless the macro was invoked in AMODE 24 in which case the recovery routine AMODE will be 31-bit.
- The AMODE at the retry point will be the same as the AMODE on entry to the recovery routine.

Various mode considerations: Depending on address space, cross-memory (the primary, secondary, and home address spaces are the same), and access register (AR) modes, you need to select the proper ESTAE type as follows:

- If your program is to execute in 31-bit addressing mode, you must use the SP Version 2 of the ESTAE macro or a later version.
- Callers that are in primary address space control (ASC) mode and not in cross-memory mode can issue either ESTAE or ESTAEX.
- Callers that are in access register (AR) mode or in cross-memory mode must use ESTAEX.
- IBM recommends that all callers use the ESTAEX macro, unless your program and your recovery routine are in 24-bit addressing mode, in which case you need to use ESTAE.

Depending on whether you code ESTAE or ESTAEX, the system passes the address of the user-specified parameter list differently. The SDWAPARM field in the SDWA contains either the address of the parameter list (ESTAE), or the address of a doubleword that contains the address and ALET of the parameter list (ESTAEX). When you run in AMODE 64 (as indicated by specifying AMODE64=YES through the SYSSTATE macro) and invoke ESTAEX, your ESTAEX routine will get control in AMODE 64. The 8-byte area pointed to by the SDWAPARM field will be the 8-byte address of the parameter area.

See the information on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about writing recovery routines.

The descriptions of ESTAE and ESTAEX are:

- The standard form of the ESTAE macro, which includes general information about the ESTAE and ESTAEX macros, with some specific information about the ESTAE macro. The syntax of the ESTAE macro is presented, and all ESTAE parameters are explained.
- The standard form of the ESTAEX macro, which includes information specific to the ESTAEX macro. The syntax of the ESTAEX macro is presented.
- The list form of the ESTAE and ESTAEX macros.
- The execute form of the ESTAE and ESTAEX macros.

Note: The ESTAE and ESTAEX macros have the same environment specifications, register information, programming requirements, restrictions and limitations, and performance implications described as follows, except where noted in the explanation for ESTAEX.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key. To use the CANCEL, BRANCH, KEY, or TOKEN parameters, one of the following: • Supervisor state • PKM allowing key 0-7 (for BRANCH, key 0 only) • APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

If the program is in AR mode, you must use ESTAEX rather than ESTAE; issue the SYSSTATE macro with the ASCENV=AR parameter before you issue ESTAEX. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

For Branch-entry, IBM recommends that you have no EUT FRRs.

IBM recommends that you do not use the ESTAE or ESTAEX macro to deactivate and no longer define a FESTAE recovery routine that was defined and activated by a FESTAE macro.

Input register information

Before issuing the ESTAE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code if GPR 15 contains X'4'; otherwise, used as a work register by the system

1
Used as a work register by the system

2
If you specify KEY=SAVE, used as a work register by the system; otherwise, unchanged

3-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the ESTAE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAE.
ESTAE	
ƀ	One or more blanks must follow ESTAE.
exit addr	*exit addr*: A-type address, or register (2) - (12).
0	
,CT	**Default**: CT
,OV	
,PARAM=*list addr*	*list addr*: A-type address, or register (2) - (12).
,XCTL=NO	**Default**: XCTL=NO
,XCTL=YES	

Syntax	Description
,PURGE=NONE	**Default**: PURGE=NONE
,PURGE=QUIESCE	
,PURGE=HALT	
,ASYNCH=YES	**Default**: ASYNCH=YES
,ASYNCH=NO	
,CANCEL=YES	**Default**: CANCEL=YES
,CANCEL=NO	
,TERM=NO	**Default**: TERM=NO
,TERM=YES	
,BRANCH=NO	**Default**: BRANCH=NO
,BRANCH=YES,SVEAREA=*save*	*save addr*: A-type address, or register (2) - (12) or (13).
addr	
,KEY=SAVE	*storage key*: Any numeral in the range 0-15.
,KEY=*storage key*	
,RECORD=NO	**Default**: RECORD=NO
,RECORD=YES	
,TOKEN=*token addr*	*token addr*: A-type address, or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SDWALOC31=NO	**Default**: SDWALOC31=NO
,SDWALOC31=YES	
,SPIEOVERRIDE=NO	**Default**: SPIEOVERRIDE=NO
,SPIEOVERRIDE=YES	

Parameters

The parameters are explained as follows.

exit addr
0
Specifies the 31-bit address of an ESTAE recovery routine to be entered if the task issuing this macro ends abnormally. If you specify 0, the most recent ESTAE recovery routine is deactivated and no longer defined.

The ESTAEX exit always gets control in 31-bit mode, regardless of the mode in which the macro was invoked.

,CT
,OV
Specifies that a new ESTAE recovery routine is to be defined and activated (CT), or indicates that parameters passed in this ESTAE macro are to overlay the data contained in the previous ESTAE routine (OV).

,PARAM=*list addr*
Specifies the 31-bit address of a user-defined list containing data to be used by the ESTAE routine when it is scheduled for execution.

,XCTL=NO
,XCTL=YES
Specifies that the ESTAE recovery routine will be deactivated and no longer defined (NO) or will remain activated and defined (YES) if this program issues an XCTL macro.

,PURGE=NONE
,PURGE=QUIESCE
,PURGE=HALT
Specifies that all outstanding requests for I/O operations are not to be saved when the ESTAE routine receives control (HALT), or that I/O processing is to be allowed to continue normally when the ESTAE routine receives control (NONE), or that all outstanding requests for I/O operations are to be saved when the ESTAE routine receives control (QUIESCE). If QUIESCE is specified, the user's retry routine can restore the outstanding I/O requests.

For PURGE=QUIESCE and PURGE=HALT, RTM requests that all I/O be purged at the task level for the current task. Be aware that the purge request involves all I/O started by the task, not just the I/O started by the program that created this recovery routine. PURGE=QUIESCE must thus be used carefully, as it may wait for I/O that was not started by the program that created this recovery routine. Likewise, PURGE=HALT must be used carefully as it may terminate I/O that was not started by the program that created this recovery routine.

PURGE=NONE specifies that all control blocks affected by input/output processing can continue to change during ESTAE routine processing. If you specify PURGE=NONE and the error was an error in input/output processing, recursion develops when an input/output interruption occurs, even if the ESTAE routine is in progress. Thus, it will appear that the ESTAE routine failed when, in reality, input/output processing caused the failure.

Note:

1. You need to understand PURGE processing before using this parameter. For information about PURGE processing, see *z/OS DFSMSdfp Advanced Services*.
2. When using PURGE, you need to consider any access-method ramifications. See the appropriate DFP information for the particular access method you are using to determine these ramifications.
3. The system performs the requested I/O processing only for the first ESTAE-type recovery routine that gets control. Subsequent routines that get control receive an indication of the I/O processing previously done, but no additional processing is performed.

,ASYNCH=YES
,ASYNCH=NO
Specifies that asynchronous exit processing will be allowed (YES) or prohibited (NO) while the user's ESTAE is running.

ASYNCH=YES must be coded if:

- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the ESTAE routine.
- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE=NONE is specified and the ESTAE routine issues the CHECK macro for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: If ASYNCH=YES is specified and the error was an error in asynchronous exit handling, recursion will develop when an asynchronous exit handling was the cause of the failure.

,CANCEL=YES
,CANCEL=NO

Specifies whether you want to allow the recovery routine to be interrupted by cancel or detach processing.

To allow a recovery routine to be interrupted, specify CANCEL=YES.

To prevent a recovery routine from being interrupted, specify CANCEL=NO. If a cancel or detach is attempted against a recovery routine for which you have specified CANCEL=NO, MVS defers cancel and detach processing until the recovery routine returns control to the system.

Note:

1. If a recovery routine that runs under the CANCEL=NO option can be called by an unauthorized program running under the same task, IBM recommends that you specify ASYNCH=NO for each ESTAE(X) macro that the recovery routine issues. This also includes any ESTAE(X) macros issued by programs that the recovery routine calls.
2. If a recovery routine running under the CANCEL=NO option calls an unauthorized program, cancel and detach processing is also deferred for the called program.

,TERM=NO
,TERM=YES

Specifies that the ESTAE routine will be scheduled (YES) or will not be scheduled (NO) in the following situations:

- System-initiated logoff
- Job step timer expiration
- Wait time limit for job step exceeded
- DETACH macro without the STAE=YES parameter issued from a higher-level task (possibly by the system if the higher-level task encountered an error)
- Operator cancel
- Error on a higher level task
- Error in the job step task when a nonjob step task issued the ABEND macro with the STEP parameter.
- z/OS UNIX is canceled and the user's task is in a wait in the z/OS UNIX kernel.

When the ESTAE routine is entered because of one of the preceding reasons, retry is not permitted. If a dump is requested at the time the ABEND macro is issued, it is taken before entry into the ESTAE routine.

Note: If DETACH was issued with the STAE parameter, the following occurs for the task to be detached:

- All ESTAE routines are entered.
- The most recently activated STAE routine is entered.
- All STAI/ESTAI routines are entered unless one of the STAI routines issues return code 16.

In these cases, entry to the routine occurs before dumping and retry is not permitted.

,BRANCH=NO
,BRANCH=YES,SVEAREA=*save addr*
Specifies that an SVC entry to the ESTAE service routine is to be performed (NO) or that a branch entry is to be performed (YES). The save area is a 72-byte area used to save the general registers. If the caller is not in key zero, the KEY parameter must be specified.

BRANCH and SVEAREA are not valid on ESTAEX.

,KEY=SAVE
,KEY=*storage key*
Specifies that supervisor state users who are not in key zero can use the branch entry interface to the ESTAE service routine.

If the user specifies KEY=SAVE, the macro saves the current PSW protection key in register 2 and issues a set protection key instruction (SPKA) to change to protection key zero. When the ESTAE service routine returns control, it restores the original PSW key from register 2. Therefore, the user should save register 2 before the macro expansion and restore it afterwards. Specifying KEY=SAVE destroys the contents of register 2 during the macro expansion.

On the other hand, if the user knows the current PSW protection key, he may specify it directly in the form KEY=(0-15) to eliminate saving and restoring the original protection key. This procedure eliminates an IPK instruction and prevents the use of register 2 in the macro expansion.

KEY is not valid on ESTAEX. KEY is optional and valid only with BRANCH=YES,SVEAREA=*save addr*.

,RECORD=NO
,RECORD=YES
Specifies whether the system diagnostic work area (SDWA) is to be recorded in SYS1.LOGREC. If you specify RECORD=YES, the system records the entire SDWA (including the fixed length base, the variable length recording area, and the recordable extensions) in SYS1.LOGREC when the associated ESTAE recovery routine returns control, unless the recovery routine indicates otherwise by issuing the SETRP macro with RECORD=NO.

If you specify RECORD=NO, the system does not record the SDWA in SYS1.LOGREC, unless the recovery routine indicates otherwise by issuing the SETRP macro with RECORD=YES.

,TOKEN=*token addr*
Specifies that a four-byte token is to be associated with the ESTAE routine. Unauthorized or accidental destruction of the ESTAE routine is prevented because the ESTAE cannot be canceled or overlaid unless the same token is specified.

With CT: ESTAE processing places the token created for this request in the location specified by *token addr* as well as in the ESTAE parameter list.

With OV: ESTAE processing locates the specified ESTAE routine for the current RB and replaces the routine information. If there are any newer ESTAE routines for the RB, they are deactivated and no longer defined.

With a recovery routine address of 0: ESTAE processing locates the specified ESTAE routine for the current RB and deactivates the routine. The routine is no longer defined. Any newer ESTAE routines for the RB are deactivated and no longer defined.

,RELATED=*value*
Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and content of the information specified are at the discretion of the user, and may be any valid coding values.

,SDWALOC31=NO
,SDWALOC31=YES
Specifies that the SDWA be in 31-bit storage (YES) or the default 24-bit storage (NO). You must specify SDWALOC31=YES when the your program is running in AMODE 31 and you are using 64-bit general purpose registers, because the time-of-error 64-bit GPRs are only presented to routines with an SDWA in 31-bit storage. Only routines with an SDWA in 31-bit storage can retry while setting those registers.

Note: The SDWALOC31= parameter applies to ESTAE only. (For ESTAEX, the SDWA is always in 31-bit storage.)

,SPIEOVERRIDE=NO
,SPIEOVERRIDE=YES

SPIEOVERRIDE specifies that the ESTAEX recovery exit must receive control for all program exceptions even if a SPIE or ESPIE exit is established.

While the recovery routine that requests this parameter is established, no SPIE or ESPIE exit can receive control.

You can use this parameter to ensure that the ESTAEX recovery exit receives control for all program exceptions that occur while running in Problem state.

The SPIEOVERRIDE parameter is not required for programs that run in Supervisor state, run in cross-memory, or hold any lock, because SPIE and ESPIE exits are not eligible to receive control in these environments.

SPIEOVERRIDE is not valid on ESTAE.

The default value is SPIEOVERRIDE=NO.

ABEND codes

None.

Return and reason codes

When control returns to the instruction following the ESTAE macro, GPR 15 contains one of the following return codes and GPR 0 contains one of the following reason codes.

Table 12. Return and Reason Codes for the ESTAE Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: Successful completion of the ESTAE request. **Action**: None.
04	00	**Meaning**: Program error. ESTAE OV was specified but ESTAE CT was performed. No valid ESTAE recovery routine existed.
04	04	**Meaning**: Program error. ESTAE OV was specified but ESTAE CT was performed. The last ESTAE recovery routine was not owned by the user's RB. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.
04	08	**Meaning**: Program error. ESTAE OV was specified but ESTAE CT was performed. The last ESTAE recovery routine was not created at the current linkage stack level. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.
04	0C	**Meaning**: Program error. ESTAE OV was specified but ESTAE CT was performed. The last recovery routine was not an ESTAE recovery routine. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.

Table 12. Return and Reason Codes for the ESTAE Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0C	None	**Meaning**: Program error. A recovery routine address equal to zero was specified, and either • There are no recovery routines for this TCB, • The most recent recovery routine is not owned by the caller, • The most recent recovery routine is not an ESTAE recovery routine, or • The ESTAE was created with the TOKEN parameter and on a deactivate request, either – The token was not specified or – The token does not match. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.
10	None	**Meaning**: System error. An unexpected error was encountered while this request was being processed. **Action**: Rerun your program one or more times. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel.
14	None	**Meaning**: Environmental error. ESTAE was unable to obtain storage for a system data area. **Action**: Free some storage and reissue the ESTAE macro.
18	None	**Meaning**: Program error. ESTAE OV request was invalid for one of the following reasons: • ESTAE OV with the TOKEN parameter was specified but – No ESTAE recovery routine exists or – The recovery routine is not an ESTAE recovery routine created with the matching token value by the current RB. • ESTAE OV without the TOKEN parameter was specified but the ESTAE recovery routine was created with the TOKEN parameter. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.
1C	None	**Meaning**: Program error. ESTAE was unable to access the input parameter list. **Action**: Make sure the parameter list is in the primary address space and reissue the ESTAE macro.
20	None	**Meaning**: Program error. XCTL=YES was rejected because the linkage stack was not at the same level as it was when the RB was created. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.
24	None	**Meaning**: Program error. A recovery routine address equal to zero was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.
28	None	**Meaning**: Program error. ESTAE OV was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level. **Action**: Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate.
30	None	**Meaning**: Program error. Branch-entered ESTAE CT was specified, but it was rejected because the caller has a cross-memory environment. **Action**: Use ESTAEX for programs that run in a cross-memory environment.

Example 1

If an error occurs, pass control to the ESTAE routine specified by register 4, allow asynchronous exit processing, do not allow special error processing, do not branch enter, and default to CT and PURGE=NONE.

```
ESTAE (4),ASYNCH=YES,TERM=NO,BRANCH=NO
```

Example 2

If an error occurs, pass control to the ESTAE routine specified by register 4. The address of the ESTAE parameter list is in register 2. Place the token associated with this ESTAE routine in TOKENFLD.

```
ESTAE (4),PARAM=(2),TOKEN=TOKENFLD
```

Example 3

If an error occurs, pass control to the ESTAE routine labeled ADDR, allow synchronous exit processing, halt I/O, allow special error processing, branch enter, use the 72-byte save area at SADDR, and execute the execute form of the macro. EXEC is the label of the ESTAE parameter list built by a list form of the macro elsewhere in this program.

```
ESTAE ADDR,ASYNCH=YES,PURGE=HALT,TERM=YES,BRANCH=YES,   X
      SVEAREA=SADDR,MF=(E,EXEC)
```

Example 4

Request an overlay of the existing ESTAE recovery routine with the following options: the address of the parameter list is at PLIST, I/O will be halted, no asynchronous exits will be taken, ownership will be transferred to the new request block resulting from any XCTL macros.

```
ESTAE   ADDR,OV,PARAM=PLIST,XCTL=YES,PURGE=HALT,ASYNCH=NO
```

Example 5

Provide the pointer to the recovery code in the register called EXITPTR, place the address of the ESTAE parameter list in register 9. Register 8 points to the area where the ESTAE parameter list (created with the MF=L option) was moved.

```
ESTAE   (EXITPTR),PARAM=(9),MF=(E,(8))
```

ESTAEX - Specify task abnormal exit extended

Note: The ESTAEX macro has the same environment specifications, register information, programming requirements, restrictions and limitations, and performance implications as the ESTAE macro, with the exceptions that follow.

Environment

The requirements for the caller of ESTAEX that are different from ESTAE are:

Environmental factor	Requirement
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary or access register (AR)

Programming requirements

If the program is in AR mode:

- Issue the SYSSTATE macro with the ASCENV=AR parameter before you issue ESTAEX. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.
- User parameters, specified on the PARAM parameter, can be located in any address space.

Restrictions

The caller of ESTAEX cannot have an EUT FRR established.

The parameters on the standard form of the ESTAEX macro are the same as for the standard form of the ESTAE macro, except BRANCH, SVEAREA, and KEY, which are not valid for ESTAEX.

Syntax

The standard form of the ESTAEX macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAEX.
ESTAEX	
ƀ	One or more blanks must follow ESTAEX.
exit addr	*exit addr*: A-type address, or register (2) - (12).
0	
,CT	**Default**: CT
,OV	
,PARAM=*list addr*	*list addr*: A-type address, or register (2) - (12).
,XCTL=NO	**Default**: XCTL=NO
,XCTL=YES	
,PURGE=NONE	**Default**: PURGE=NONE
,PURGE=QUIESCE	
,PURGE=HALT	

Syntax	Description
,ASYNCH=YES	**Default**: ASYNCH=YES
,ASYNCH=NO	
,CANCEL=YES	**Default**: CANCEL=YES
,CANCEL=NO	
,TERM=NO	**Default**: TERM=NO
,TERM=YES	
,RECORD=NO	**Default**: RECORD=NO
,RECORD=YES	
,TOKEN=*token addr*	*token addr*: A-type address, or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SPIEOVERRIDE=NO	**Default**: SPIEOVERRIDE=NO
,SPIEOVERRIDE=YES	

Parameters

The parameters are explained under the syntax for the standard form of the ESTAE macro.

ABEND codes

None.

Return and reason codes

When control returns to the instruction following the ESTAEX macro, the return code in GPR 15 and the reason code in GPR 0 might be different from those for the ESTAE macro. The following table lists the return and reason codes for ESTAEX.

Table 13. Return and Reason Codes for the ESTAEX Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: Successful completion of the ESTAEX request. **Action**: None.
04	00	**Meaning**: Program error. ESTAEX OV was specified but ESTAEX CT was performed. No valid ESTAE recovery routine existed. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.

Table 13. Return and Reason Codes for the ESTAEX Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
04	04	**Meaning**: Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last ESTAE recovery routine was not owned by the user's RB. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
04	08	**Meaning**: Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last ESTAE recovery routine was not created at the current linkage stack level. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
04	0C	**Meaning**: Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last recovery routine was not an ESTAE recovery routine. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
08	None	**Meaning**: Program error. The ESTAEX request was not valid. **Action**: Correct the request and either reissue the ESTAEX macro or rerun your program, as appropriate.
0C	None	**Meaning**: Program error. A recovery routine address equal to zero was specified, and either • There are no recovery routines for this TCB, • The most recent recovery routine is not owned by the caller, • The most recent recovery routine is not an ESTAE recovery routine, or • The ESTAE was created with the TOKEN parameter and on a deactivate request, either – The token was not specified or – The token does not match. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
10	None	**Meaning**: System error. An unexpected error was encountered while this request was being processed. **Action**: Rerun your program one or more times. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel.
14	None	**Meaning**: Environmental error. ESTAEX was unable to obtain storage for a system data area. **Action**: Free some storage and reissue the ESTAEX macro.
18	None	**Meaning**: Program error. ESTAEX OV was requested and one of the following occurred: • The TOKEN parameter was specified and the ESTAE recovery routine is not owned by the current RB • The TOKEN parameter was not specified but the ESTAE recovery routine was created with the TOKEN parameter. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
1C	None	**Meaning**: Program error. ESTAEX was unable to access the input parameter list. **Action**: Make sure the parameter list is contained in the primary address space and reissue the ESTAEX macro.

Table 13. Return and Reason Codes for the ESTAEX Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
20	None	**Meaning**: Program error. XCTL=YES was rejected because the linkage stack was not at the same level as it was when the RB was created. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
24	None	**Meaning**: Program error. A recovery routine address equal to zero was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
28	None	**Meaning**: Program error. The caller was disabled. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
2C	None	**Meaning**: Program error. The caller was locked. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
30	None	**Meaning**: Program error. The caller had FRRs on the current FRR stack. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.
34	None	**Meaning**: Program error. The caller was in SRB mode. **Action**: Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate.

Example

The following example show how to establish an ESTAEX recovery routine that receives control for all abends, including CANCEL or DETACH abends, and overrides any SPIE or ESPIE exit that is established:

```
ESTAEX  addr,PARM=parmaddr,TERM=YES,SPIEOVERRIDE=YES
```

ESTAE and ESTAEX - List form

The list form of ESTAE or ESTAEX is used to construct a remote control parameter list.

Syntax

The list form of ESTAE or ESTAEX is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAE or ESTAEX.
ESTAE	
ESTAEX	

Syntax	Description
ƀ	One or more blanks must follow ESTAE or ESTAEX.
exit addr	*exit addr*: A-type address.
,PARAM=*list addr*	*list addr*: A-type address.
,PURGE=NONE	**Default**: PURGE=NONE
,PURGE=QUIESCE	
,PURGE=HALT	
,ASYNCH=YES	**Default**: ASYNCH=YES
,ASYNCH=NO	
,CANCEL=YES	**Default**: CANCEL=YES
,CANCEL=NO	
,TERM=NO	**Default**: TERM=NO
,TERM=YES	
,RECORD=NO	**Default**: RECORD=NO
,RECORD=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SDWALOC31=NO	**Default**: SDWALOC31=NO
,SDWALOC31=YES	**Note:** SDWALOC31 is supported only by ESTAE.
,SPIEOVERRIDE=NO	**Default**: SPIEOVERRIDE=NO
,SPIEOVERRIDE=YES	
,MF=L	

Parameters

The parameters are explained under the standard form of the ESTAE or ESTAEX macro with the following exception:

,MF=L
Specifies the list form of the ESTAE or ESTAEX macro.

ESTAE or ESTAEX - Execute form

A remote control parameter list is used in, and can be modified by, the execute form of the ESTAE or ESTAEX macro. The control parameter list can be generated by the list form of the ESTAE or ESTAEX macro. Any combination of exit addr, PARAM, XCTL, PURGE, ASYNCH, TERM, RECORD, TOKEN, and SPIEOVERRIDE can be specified to dynamically change the contents of the remote ESTAE or ESTAEX parameter list. If the TOKEN parameter was previously specified and is to be used again without change, TKNPASS=YES must be coded. Any fields not specified on the macro remain as they were before the current ESTAE or ESTAEX request was made.

Note: To ensure that the ESTAE or ESTAEX parameters are correct, the control parameter list specified for the execute form of the ESTAE and ESTAEX macros must be initialized from a list form of the macro.

Syntax

The execute form of the ESTAE or ESTAEX macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAE or ESTAEX.
ESTAE	
ESTAEX	
ƀ	One or more blanks must follow ESTAE or ESTAEX.
exit addr	*exit addr*: RX-type address, or register (2) - (12).
0	
,CT	
,OV	
,PARAM=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,XCTL=NO	
,XCTL=YES	
,PURGE=NONE	
,PURGE=QUIESCE	
,PURGE=HALT	

Syntax	Description
,ASYNCH=YES	
,ASYNCH=NO	
,CANCEL=YES	**Default**: CANCEL=YES
,CANCEL=NO	
,TERM=NO	
,TERM=YES	
,BRANCH=NO	**Note**: BRANCH and SVEAREA are not valid on ESTAEX.
,BRANCH=YES,SVEAREA=*save*	*save addr*: RX-type address, or register (2) - (12) or (13).
addr	
,KEY=SAVE	*storage key*: Any numeral in the range 0-15.
,KEY=*storage key*	**Note**: KEY is not valid on ESTAEX.
,RECORD=NO	
,RECORD=YES	
,TOKEN=*token addr*	*token addr*: RX-type address, or register (2) - (12).
,TKNPASS=NO	**Default**: TKNPASS=NO
,TKNPASS=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SDWALOC31=NO	**Default**: SDWALOC31=NO
,SDWALOC31=YES	**Note:** SDWALOC31 is supported only by ESTAE.
,SPIEOVERRIDE=NO	**Default**: SPIEOVERRIDE=NO
,SPIEOVERRIDE=YES	
,MF=(E,*ctrl addr)*	*ctrl addr*: RX-type address, or register (1) or (2) - (12).

Parameters

The parameters are explained under the standard form of the ESTAE or ESTAEX macro, with the following exceptions:

,TKNPASS=NO
,TKNPASS=YES
Specifies that a previously-specified token, indicated in the parameter list, should be ignored (NO), or should remain part of the specification (YES).

,MF=(E,*ctrl addr)*
Specifies the execute form of the ESTAE or ESTAEX macro using a remote control parameter list.

,ASYNCH=NO
,ASYNCH=YES
Asynchronous exit processing will not be (NO) or will be prohibited (YES) while the user's ARR is running through the IEAARR service.

Chapter 7. ETCON — Connect entry table

Description

The ETCON macro connects one or more previously created entry tables to the specified linkage table indexes in the current home address space. If an entry table is connected to a system linkage index (an index reserved with the SYSTEM=YES option of the LXRES macro), the entry table is connected to the linkage table of every address space, both present and future.

The connection created by the ETCON macro remains in effect until one of the following occurs:

- The ETDIS macro removes the connection.
- The entry table owner terminates.
- The address space to which the table is connected terminates unless the connection was to a system linkage index.
- The system is re-IPLed.

Related macros

ETDEF, ETCRE, ETDES, and ETDIS

Environment

The requirements for callers of ETCON are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt Status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	The parameter list passed to the ETCON macro must be addressable in primary mode at the time the macro is issued.

Programming requirements

None.

Restrictions

The restrictions on the use of the ETCON macro are the following:

- If an entry table contains entries that cause address space switches, the entry table owner must have PT and SSAR authorization to issue PT and SSAR instructions to the home address space.
- An entry table can be connected only once to a single linkage table.
- The linkage index and the entry table being connected must be owned by the same task (the cross memory resource owning task of the home address space).

Any violation of these restrictions causes the system to abnormally end the calling program.

Input register information

The ETCON macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter:

- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETCON macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.
- Otherwise, the caller must ensure that the following general purpose register contains the specified information:

Register
Contents

13
The address of an 18-word save area

Output register information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When using the standard form of ETCON, do not use register 2 as your program's base register. The macro modifies register 2 and then uses a branch instruction. Register 2 is restored by the time control returns to your program.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

Performance implications

None.

Syntax

The ETCON macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETCON.

Syntax	Description
ETCON	
ƀ	One or more blanks must follow ETCON.
TKLIST=*addr*	*addr*: RX-type address or register (0) - (12).
,LXLIST=*lx list addr*	*lx list addr*: RX-type address or register (2) - (12).
,ELXLIST=*elx list addr*	*elx list addr*: RX-type address or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

TKLIST=*addr*
Specifies the address of a list of fullword tokens representing the entry tables to be connected to the linkage table. The first entry in the list must be the number of tokens that follow (from 1 to 32). The tokens are the values returned in register 0 when the ETCRE macro is issued.

,LXLIST=*addr*
,ELXLIST=*addr*
lx list addr specifies the address of a list of linkage index (LX) values to which the specified entry tables are to be connected. The list contains fullword entries, the first of which must be the number of linkage index values that follow (from 1 to 32). The number of linkage index values must be the same as the number of tokens. The first entry table is connected to the first linkage index; the second entry table is connected to the second linkage index, and so on.

elx list addr specifies the address of an area that contains extended linkage index (LX) values to which the specified entry tables are to be connected. The first word in the area must be the number of extended LX values that follow (from 1 to 32). Each subsequent eight bytes contains an extended LX value which consists of a 4-byte sequence number followed by an LX value. The number of extended linkage index values must be the same as the number of tokens. The first entry table is connected to the first linkage index; the second entry table is connected to the second linkage index, and so on. If the sequence number in the entry is incorrect, the system issues abend X'052' with reason code X'051B'.

,RELATED=*value*
Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and can be any valid coding values.

ABEND codes

052
053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

When ETCON macro returns control to your program, GPR 15 contains a return code.

Table 14. Return Code for the ETCON Macro	
Hexadecimal Return Code	**Meaning**
00	**Meaning**: The specified connections were successfully made. **Action**: None required.

Examples

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in *z/OS MVS Programming: Extended Addressability Guide*.

ETCON - List form

The list form of the ETCON macro constructs a nonexecutable parameter list. This list, or a copy of it for reentrant programs, can be referred to by the execute form of the macro.

Syntax

The list form of the ETCON macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETCON.
ETCON	
ƀ	One or more blanks must follow ETCON.
TKLIST=*addr*	*addr*: A-type address.
,LXLIST=*addr*	*addr*: A-type address.
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

Parameters

The parameters are explained under the standard form of the ETCON macro, with the following exception:

,MF=L
Specifies the list form of the ETCON macro.

ETCON - Execute form

The execute form of the ETCON macro can refer to and modify a remote parameter list created by the list form of the macro.

Syntax

The execute form of the ETCON macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETCON.
ETCON	
ƀ	One or more blanks must follow ETCON.
TKLIST=*addr*	*addr*: RX-type address or register (0) - (12).
,LXLIST=*addr*	*addr*: RX-type address or register (0) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=(E,*cntl addr*)	*cntl addr*: RX-type address or register (0) - (12).

Parameters

The parameters are explained under the standard form of the ETCON macro with the following exception:

,MF=(E,*cntl addr*)
Specifies the execute form of the ETCON macro. This form uses a remote parameter list.

Chapter 8. ETCRE — Create entry table

Description

The ETCRE macro builds a program-call entry table based upon descriptions of each entry. A token representing the created entry table is returned to the requestor. You must use this token in all subsequent references to the entry table.

Related macros

ETDEF, ETDES, ETCON, and ETDIS

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in primary address space

Programming requirements

Before issuing ETCRE, the caller must create the ETD parameter list that ETCRE uses as input. The parameter list defines the names and characteristics of the program call (PC) routines that the entry table will define. To create the parameter list, the caller can issue the ETDEF macro or can code the data constants needed to define the list. If data constants are coded, the caller can use mapping macro IHAETD to map them.

The created entry table is owned by the cross memory resource ownership task in the current home address space. When the cross memory resource ownership task terminates, entry tables are disconnected and freed.

Note: Programs written before SP/Version 3, which use data constants to define the parameter list (the resulting ETD was called a format 0 ETD) and which use IHAETD to map the data area, will still work. For information about the format 0 ETD, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Restrictions

None.

Input register information

The ETCRE macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter

- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETCRE macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.
- Otherwise, the caller must ensure that the following general purpose register contains the specified information:

Register
Contents

13
The address of an 18-word save area

Output register information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
The 32-bit token associated with the new entry table

1
Used as a work register by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Performance implications

None.

Syntax

The ETCRE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETCRE.
ETCRE	
ƀ	One or more blanks must follow ETCRE.

Syntax	Description
ENTRIES=*addr*	*addr*: RX-type address or register (0) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

ENTRIES=*addr*
Specifies the address of the parameter list that defines the PC routines.

An entry index value that does not have a description results in an invalid entry in the entry table. If the program name field in an ETD entry contains zeros, an invalid entry is created for that entry index. A program call to an invalid entry causes the caller to be abnormally terminated. The ETCRE caller is abnormally terminated if any of the reserved fields are nonzero or if the system cannot locate the specified program name.

,RELATED=*value*
Specifies information used to self-document macros by relating functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

ABEND codes

052
053

See *z/OS MVS System Codes* for an explanation and programmer responses for this code.

Return codes

When ETCRE macro returns control to your program, GPR 15 contains a return code.

Table 15. Return Code for the ETCRE Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: The entry table is successfully created. **Action**: None required.

Example

Show the relationship between the ETCRE and the ETDEF macros. ETDEF builds an entry table descriptor (ETD) that contains two ETD entries. The first entry, associated with PROGRAM1, is for a PC routine that runs in supervisor state. The second entry, associated with PROGRAM2, is for a PC routine that runs in problem state.

```
*
* CREATE THE ENTRY TABLE
*
         .
         .
         LA    2,ETSTART
         ETCRE ENTRIES=(2)
         .
         .
*
```

```
* DEFINE START OF ETD
*
ETSTART  ETDEF TYPE=INITIAL              START ETD
*
* DEFINE ENTRIES
*
ETEX2    ETDEF TYPE=ENTRY,PROGRAM='PROGRAM1',AKM=(0:15)
         ETDEF TYPE=ENTRY,PROGRAM='PROGRAM2',AKM=(0:7)
*
* DEFINE END OF ETD
*
        ETDEF TYPE=FINAL
```

Chapter 9. ETDEF — Create an entry table descriptor (ETD)

Description

The ETDEF macro builds and modifies the parameter that the ETCRE macro uses to build an entry table. The parameter, called the entry table descriptor (ETD), consists of a header, followed by one or more entries, called ETD entries, each one describing a PC routine. The address of the ETD is input to the ENTRIES parameter on the ETCRE macro.

The TYPE parameter on the ETDEF macro determines which process the ETDEF macro is to perform:

- ETDEF TYPE=INITIAL generates the header for the ETD. (Issue this macro once for each ETD.)
- ETDEF TYPE=ENTRY generates one ETD entry. (You can issue this macro up to 256 times for each ETD.)
- ETDEF TYPE=FINAL terminates the ETD. (Issue this macro once for each ETD.)
- ETDEF TYPE=SET,ETEADR replaces the variable fields of an existing ETD entry.
- ETDEF TYPE=SET,HEADER changes the number of entries in an existing ETD header.

Related macros

ETDES, ETCRE, ETCON, and ETDIS

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or Supervisor state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Serialization:	Not applicable
Interrupt status:	None
Locks:	None
Control parameters:	None

Programming requirements

You need to create an ETD at compile time through TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL parameters and initialize the information for the entries at execution time through TYPE=SET,ETEADR. Therefore, ETDEF with the TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL parameters works like a list form of the macro. However, unlike the execute form of a macro, which changes only the values you specify, the TYPE=SET form of ETDEF completely *replaces* the variable fields of an ETD entry, taking the default values for any parameters you omit, and leaves constant fields as initialized. This information describes the two forms separately.

Although ETDEF is the preferred programming interface, if you have an existing ETD and you want to update the parameters (for example, change the user parameter), you might choose to use the IHAETD

mapping macro instead of ETDEF. If you change an existing ETD, without using any of the function of MVS/SP Version 3, you can use IHAETD with the format number of “0”. The format of IHAETD is in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) under "ETD".

Note: When changing code to use ETDEF in place of the IHAETD mapping macro, be sure to specify PC=BASIC so that the PC does not become a stacking PC. If you want to change an existing PC routine to a stacking PC, be sure to change the PT instruction in the PC routine to a PR.

Restrictions

None.

Register information

The ETDEF macro does not use any registers, except for those you use to specify parameters.

Performance implications

None.

TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL parameters

The ETDEF macro with the TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL options works like a list form of a macro.

Syntax for TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL

This form is described as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETDEF.
ETDEF	
ƀ	One or more blanks must follow ETDEF.
TYPE=INITIAL	**Valid Parameters**: RELATED
TYPE=ENTRY	**Required Parameters**: PROGRAM or ROUTINE, AKM
	EKM, ARR, ASCMODE, EAX, EK, PARM1, PARM2, PC, PKM, SASN, SSWITCH, STATE, RELATED, ASYNCH, CANCEL
TYPE=FINAL	RELATED
,AKM=*key-list*	*key-list*: List of keys or key ranges where a key is a number 0 - 15.

Syntax	Description
,ARR=*arr*	*arr*: A-type address, or alphanumeric character string enclosed by single quotation marks.
,ARRCOND=NO	**Default**: ARRCOND=NO
,ARRCOND=YES	Valid only when ARR is also coded.
,ASYNCH=YES	**Default**: ASYNCH=YES
,ASYNCH=NO	Valid only when ARR is also coded.
,CANCEL=YES	**Default**: CANCEL=YES
,CANCEL=NO	Valid only when ARR is also coded.
,ASCMODE=PRIMARY	**Default**: ASCMODE=PRIMARY
,ASCMODE=AR	
,EAX=*eax-value*	*eax-value*: Half-word decimal digit.
,EK=*entry-key*	*entry-key*: Decimal digit 0 - 15.
,EKM=*key-list*	*key-list*: List of keys or key ranges where a key is a number 0 - 15.
Note: EKM is required with PKM=REPLACE.	
,PARM1=*user-parm1*	*user-parm1*: A-type address or string of up to 4 characters enclosed by single quotation marks.
,PARM2=*user-parm2*	*user-parm2*: A-type address or string of up to 4 characters enclosed by single quotation marks.
,PC=STACKING	**Default**: PC=STACKING
,PC=BASIC	
,PROGRAM=*pgm-name*	*pgm-name*: String of up to 8 alphanumeric characters, optionally enclosed by single quotation marks.
,ROUTINE=*rtn-addr*	*rtn-addr*: A-type address.
,PKM=OR	**Default**: PKM=OR

Syntax	Description
,PKM=REPLACE	
,RAMODE=31	**Default**: RAMODE=31
,RAMODE=24	
,RAMODE=64	
,RELATED=*value*	*value*: Any valid macro parameter specification.
,SASN=OLD	**Default**: SASN=OLD
,SASN=NEW	
,SSWITCH=NO	**Default**: SSWITCH=NO
,SSWITCH=YES	
,STATE=PROBLEM	**Default**: STATE=PROBLEM
,STATE=SUPERVISOR	

Parameters for TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL

The parameters are described as follows:

TYPE=INITIAL
Generates the header for the ETD.

TYPE=ENTRY
Generates an ETD entry. The system uses the defaults for any parameters you do not specify on the ETDEF TYPE=ENTRY macro. When you later specify ETDEF TYPE=SET, that macro initializes the **entire** ETD entry.

TYPE=FINAL
Specifies that the ETD is complete.

,AKM=*key-list*
Specifies a list of keys (0 through 15) or key ranges, optionally enclosed in parentheses, that identifies the authorized keys in which a problem program can use the PC routine. For example, AKM=(2,(3),5:8,(10:12),15) would authorize keys 2, 3, 5, 6, 7, 8, 10, 11, 12, and 15.

,ARR=*arr*
Specifies the associated recovery routine (ARR) that receives control if the stacking-PC routine abends. You can use the A-type address of the routine, or the name of the routine (an alphanumeric character string) enclosed in single quotation marks. If you use the name of the program, the program must be on the active LPA queue (FLPA or MLPA) or be in the PLPA or nucleus. The recovery routine will be entered in 31-bit mode. ARR is not valid with PC=BASIC.

,ARRCOND=NO,*ARRCOND=YES*
Specifies whether or not the ARR is conditional.

ARRCOND=NO, indicates that the ARR is not conditional, which means that the system follows the rules described in Using ARRs found in *z/OS MVS Programming: Authorized Assembler Services Guide*

with respect to recording in LOGREC error recording if the ARR is skipped. ARRCOND=YES indicates that no recording in LOGREC error recording is to occur if the ARR is skipped.

Use ARRCOND=YES to avoid having to provide two PCs, one without an ARR for use in an FRR environment, and one with an ARR for use when not in an FRR environment.

ARRCOND is valid only with ARR.

,ASYNCH=YES
,ASYNCH=NO
Specifies whether or not the ARR can be interrupted by asynchronous exits. ASYNCH=YES specifies that the ARR can be interrupted by asynchronous exits. ASYNCH=NO specifies that the ARR cannot be interrupted by asynchronous exits. ASYNCH=YES is the default. ASYNCH is valid only with ARR.

,CANCEL=YES
,CANCEL=NO
Specifies whether or not the ARR can be interrupted by CANCEL/DETACH processing. CANCEL=YES specifies that the ARR can be interrupted by CANCEL/DETACH processing. CANCEL=NO specifies that the ARR cannot be interrupted by CANCEL/DETACH processing. CANCEL=YES is the default. CANCEL is valid only with ARR. To specify CANCEL=NO, one of the following conditions must be true for the stacking PC routine protected by the ARR:

- The stacking PC routine runs in supervisor state.
- The entry key for the stacking PC routine is a system key.
- The stacking PC routine runs with a system key valid for the entry key mask that will either replace or be ORed with the PKM.

,ASCMODE=PRIMARY
,ASCMODE=AR
Specifies that the stacking PC routine will execute in primary ASC mode (ASCMODE=PRIMARY) or in AR ASC mode (ASCMODE=AR). ASCMODE=AR is not valid with PC=BASIC. ASCMODE=PRIMARY is the default.

,EAX=*eax-value*
Specifies the extended authorization index (EAX) that the stacking PC routine uses. Specify an EAX that is owned by the home address space of the issuer of the ETCRE macro. An EAX of X'0000' means the PC routine is not EAX-authorized. If EAX is not specified, the PC routine has the same EAX as the issuer of the PC instruction. EAX is not valid with PC=BASIC.

,EK=*entry-key*
Specifies the PSW key (0 through 15) that the PC routine will run in. EK is not valid with PC=BASIC. If you omit EK, the PC routine gets control in the key of the caller.

,EKM=*key-list*
Specifies a list of keys (0 through 15) or key ranges, optionally enclosed in parentheses, that identify the entry key mask (EKM). When the PC routine is invoked, the keys specified identify either the additional keys that are to be ORed into the PKM (if PKM=OR is also specified or taken as the default) or the keys that should replace the PKM (if PKM=REPLACE is specified). EKM is required when you specify PKM=REPLACE.

,PARM1=*user-parm1*
Specifies the address or character string to be placed in the first word of the latent parameter area associated with this ETD entry.

Addressability to the latent parameter area is through the current primary address space. The latent parameter address is set in general register 4 as a result of the PC instruction, although AR4 is unchanged by the PC instruction. If the PC routine runs in AR mode, set the access register corresponding to the latent parameter area to zero before the PC routine attempts to use it.

,PARM2=*user-parm2*
Specifies the address or character string to be placed in the second word of the latent parameter area associated with this ETD entry.

Addressability to the latent parameter area is through the current primary address space. The latent parameter address is set in general register 4 as a result of the PC instruction, although AR4 is unchanged by the PC instruction. If the PC routine runs in AR mode, set the access register corresponding to the latent parameter area to zero before the PC routine attempts to use it.

,PC=STACKING
,PC=BASIC
Indicates that this is a stacking PC (STACKING) or not a stacking PC (BASIC). Some parameters apply only to a stacking PC. STACKING is the default.

,PROGRAM=*pgm-name*
,ROUTINE=*rtn-addr*
Specifies the PC routine. When you specify PROGRAM, the PC routine must be on the active LPA queue (FLPA or MLPA) or be in the PLPA or nucleus. The same restriction applies also to ROUTINE, unless this is a space-switching PC or the PC is to be used only in the address space that established it. In other words, the PC routine for a space-switching PC can reside in the private area of the address space in which it will run, but the ROUTINE parameter must be used to specify it.

When you specify ROUTINE, you can indicate the AMODE of the PC routine with the RAMODE parameter. When you specify PROGRAM, the system locates the PC routine and determines its AMODE.

On TYPE=ENTRY or TYPE=SET,ETEADR, either PROGRAM or ROUTINE is required.

,PKM=OR
,PKM=REPLACE
Indicates either that the entry key mask (EKM) is ORed with the PSW key mask (PKM) or replaces the current PKM. PKM=REPLACE is not valid with PC=BASIC. PKM=OR is the default.

,RAMODE=31
,RAMODE=24
,RAMODE=64
Specifies the AMODE of the routine specified on the ROUTINE parameter. RAMODE is valid only with ROUTINE. If you specify PROGRAM rather than ROUTINE, the system locates the routine and determines its AMODE. RAMODE=31 is the default.

,RELATED=*value*
Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user and may be any valid coding values.

,SASN=OLD
,SASN=NEW
Specifies whether the stacking PC routine will execute with SASN equal to the caller's PASN (SASN=OLD), or with SASN equal to the PASN of the stacking PC routine (SASN=NEW). SASN=NEW is not valid with PC=BASIC. SASN=OLD is the default.

,SSWITCH=NO
,SSWITCH=YES
Specifies whether or not the PC routine switches address spaces. If SSWITCH=NO is specified, the PC does not switch address spaces. If SSWITCH=YES is specified, the PC routine will execute in the address space of the creator of the entry table with the authority of that address space. SSWITCH=NO is the default.

,STATE=PROBLEM
,STATE=SUPERVISOR
Specifies which state the PC routine will receive control in either problem state (PROBLEM) or supervisor state (SUPERVISOR). The default is STATE=PROBLEM.

An example of using the ETDEF macro follows the description of the TYPE=SET parameter.

TYPE=SET parameter

The ETDEF macro with the SET parameter works similarly to the execute form of a macro **with this important distinction: The TYPE=SET form totally replaces all variables in an ETD entry and takes default values for all parameters you omit**. The normal execute form of a macro changes **only** the values you specify.

Constants and reserved fields that are initialized by other TYPE= forms are not updated or changed. To create an entry table in a storage area that is not initialized (for example, one just allocated through a GETMAIN request), you must first move a complete entry table of the proper (or larger) size to that area. The formatted table will provide the constants and indexes. Then, you can use ETDEF TYPE=SET to change the required entry's variable parameters.

Syntax for TYPE=SET

The form of SET is described as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETDEF.
ETDEF	
ƀ	One or more blanks must follow ETDEF.
TYPE=SET,ETEADR=*entry-addr*	**Required Parameters**: PROGRAM or ROUTINE, AKM
	Valid Parameters: EKM, ARR, ASCMODE, EAX, EK, PARM1, PARM2, PC, PKM, RAMODE, SASN, SSWITCH, STATE, RELATED, ASYNCH, CANCEL
	entry-addr: RX-type address or register (1) - (15).
TYPE=SET,HEADER=*header-addr*	**Required Parameter**: NUMETE
	Valid Parameter: RELATED
	header-addr: RX-type address or register (1) - (15).
,AKM=*key-list*	*key-list*: List of keys or key ranges where a key is a decimal digit 0 - 15.
,ARR=*arr*	*arr*: A-type address, register (2)-(12), or alphanumeric character string, enclosed by single quotation marks.
,ARRCOND=NO	**Default**: ARRCOND=NO
,ARRCOND=YES	Valid only when ARR is also coded.

Syntax	Description
,ASYNCH=YES	**Default**: ASYNCH=YES
,ASYNCH=NO	Valid only when ARR is also coded.
,CANCEL=YES	**Default**: CANCEL=YES Valid only when ARR is also coded.
,CANCEL=NO	
,ASCMODE=PRIMARY	**Default**: ASCMODE=PRIMARY
,ASCMODE=AR	
,EAX=*eax-value*	*eax-value*: Half-word decimal digit or register (2)-(12)
,EK=*entry-key*	*entry-key*: Decimal digit 0 - 15.
,EKM=*key-list*	*key-list*: List of keys or key ranges where a key is a decimal digit 0 -15.
	Note: EKM is required with PKM=REPLACE.
,NUMETE=*nbr-of-entries*	*nbr-of-entries*: 2-byte A-type address, decimal number, or register (2)-(12).
	Note: NUMETE is required with HEADER.
,PARM1=*user-parm1*	*user-parm1*: A-type address, register (2)-(12), or string of up to 4 characters enclosed by single quotation marks.
,PARM2=*user-parm2*	*user-parm2*: A-type address, register (2)-(12), or string of up to 4 characters enclosed by single quotation marks.
,PC=STACKING	**Default**: PC=STACKING
,PC=BASIC	
,PROGRAM=*pgm-name*	*pgm-name*: String of up to 8 alphanumeric characters, optionally enclosed by single quotation marks.
,ROUTINE=*rtn-addr*	*rtn-addr*: A-type address or registers (2)-(12)
,PKM=OR	**Default**: PKM=OR
,PKM=REPLACE	
,RAMODE=31	**Default**: RAMODE=31

Syntax	Description
,RAMODE=24	
,RAMODE=64	
,RELATED=*value*	*value*: Any valid macro parameter specification.
,SASN=OLD	**Default**: SASN=OLD
,SASN=NEW	
,SSWITCH=NO	**Default**: SSWITCH=NO
,SSWITCH=YES	
,STATE=PROBLEM	**Default**: STATE=PROBLEM
,STATE=SUPERVISOR	

Parameters for TYPE=SET

The parameters are described under the TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL options, with the following exceptions:

,ARRCOND=NO,*ARRCOND=YES*
Specifies whether or not the ARR is conditional.

ARRCOND=NO, which is the default, indicates that the ARR is not conditional, which means that if the system skips the ARR because of an incorrect environment, that fact is recorded in LOGREC error recording.

ARRCOND=YES indicates that if the system skips this ARR, that fact will **not** be recorded in LOGREC error recording. Use ARRCOND=YES to avoid having to provide two PCs, one without an ARR for use in an FRR environment, and one with an ARR for use when not in an FRR environment.

ARRCOND is valid only with ARR.

,NUMETE=*nbr-of-entries*
Specifies the number of contiguous entries in the ETD. *nbr-of-entries* is a decimal value from 1 to 256. NUMETE is required with the HEADER parameter. Use it to specify the number of entries you will use. It does not change the physical size of the table, but can be less than the initial size.

TYPE=SET,ETEADR=*entry-addr*
Specifies the address of the ETD entry. ETDEF TYPE=SET,ETEADR sets all the variable fields in the ETD entry that you generated through ETDEF TYPE=ENTRY macro. ETDEF TYPE=SET,ETEADR will set the ETD entry to the parameters you specify **and to the *defaults* on all parameters you omit**. That is, the system uses the default value, not the existing value, for any parameter that you omit.

TYPE=SET,HEADER=*header-addr*
Changes the size of the ETD. Use TYPE=SET,HEADER to decrease the size of the ETD from the size you originally established on ETDEF TYPE=INITIAL.

ABEND codes

None.

Return and reason codes

None.

Example

Define an entry table that has three entries. The PC routine called PCPGM receives control from a program with PSW key authorization of 8, the PC routine named OTHERTN receives control from a program with PSW authorization keys of 0 through 15, and the third PC routine called PCRTN receives control in PSW authorization key 0. The fourth ETDEF is there to show that the number of entries can be changed with ETDEF SET. (Perhaps, because of some input parameter, only a subset of all possible PC routines are set up. On another invocation of the program, perhaps all entries would be used.) The entries use all defaults other than those on the AKM parameter.

```
MYPGM   CSECT
        BALR 12,0
        USING *,12
        LOAD  EP=PCPGM
        LR    2,0
        ETDEF TYPE=SET,HEADER=MYETDS,NUMETE=3
        ETDEF TYPE=SET,ETEADR=FIRST,ROUTINE=(2),AKM=8
        ETCRE ENTRIES=MYETDS
        RETURN
         .
         .
         .
*            DATA DEFINITIONS FOR PROGRAM
         .
MYETDS  ETDEF TYPE=INITIAL
FIRST   ETDEF TYPE=ENTRY,ROUTINE=0,AKM=8
SECOND  ETDEF TYPE=ENTRY,PROGRAM=OTHERTN,AKM=0:15
THIRD   ETDEF TYPE=ENTRY,ROUTINE=PCRTN,AKM=0
FOURTH  ETDEF TYPE=ENTRY,ROUTINE=0,AKM=0
        ETDEF TYPE=FINAL
*
*
PCRTN   DS    0H
        .
        .
*     PC ROUTINE CODE
        .
        .
        END   MYPGM
```

Note that the combination of TYPE=INITIAL, ENTRY, and FINAL is essentially the list form of the macro and TYPE=SET is the execute form.

Chapter 10. ETDES — Destroy entry table

Description

The ETDES macro is used to destroy a previously-created entry table.

Related macros

ETDEF, ETCRE, ETCON, and ETDIS

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in primary address space

Programming requirements

None.

Restrictions

An entry table can be destroyed only by the address space that owns it.

Input register information

The ETDES macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter

- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETDES macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.
- Otherwise, the caller must ensure that the following general purpose register contains the specified information:

  Register
  Contents

  13
  The address of an 18-word save area

Output register information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Performance implications

None.

Syntax

The ETDES macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETDES.
ETDES	
ƀ	One or more blanks must follow ETDES.
TOKEN=*addr*	*addr*: RX-type address or register (0) - (12).
,PURGE=NO	**Default**: PURGE=NO
,PURGE=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

TOKEN=*addr*
Specifies the address of the fullword token (returned by the ETCRE macro) associated with the entry table to be destroyed.

,PURGE=NO
,PURGE=YES
Specifies whether (YES) or not (NO) the entry table is to be disconnected from all linkage tables and then destroyed.

At the time ETDES is issued, the entry table must not be connected to any linkage tables unless PURGE=YES is coded. If any outstanding connections still exist and PURGE=YES is not coded, the entry table is not destroyed and the caller is abnormally terminated.

,RELATED=*value*
Specifies information used to self-document macros by "relating" functions or services to corresponding services. The format and contents of the information specified can be any valid coding values.

ABEND codes

052
053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

When ETDES macro returns control to your program, GPR 15 contains a return code.

Table 16. Return Codes for the ETDES Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The specified entry table was destroyed. There were no connections to linkage indexes. **Action**: None required.
04	**Meaning**: The specified entry table was destroyed. There were connections to linkage indexes, PURGE=YES was specified, and the entry table was disconnected. **Action**: None required. However, you may take some action based upon your application.

Examples

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in *z/OS MVS Programming: Extended Addressability Guide*.

ETDES - List form

The list form of the ETDES macro constructs a nonexecutable parameter list. The execute form of the macro can refer to this parameter list, or a copy of it for reentrant programs.

Syntax

The list form of the ETDES macro is written as follows:

Syntax	Description

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETDES.
ETDES	
ƀ	One or more blanks must follow ETDES.
TOKEN=*addr*	*addr*: A-type address.
,PURGE=NO	**Default**: PURGE=NO
,PURGE=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

Parameters

The parameters are explained under the standard form of the ETDES macro with the following exception:

,MF=L
Specifies the list form of the ETDES macro.

ETDES - Execute form

The execute form of the ETDES macro can refer to and modify a remote parameter list created by the list form of the macro.

Syntax

The execute form of the ETDES macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETDES.
ETDES	

Syntax	Description
ƀ	One or more blanks must follow ETDES.
TOKEN=*addr*	*addr*: RX-type address or register (0) - (12).
,PURGE=NO	**Default**: PURGE=NO
,PURGE=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=(E,*cntl addr*)	*cntl addr*: RX-type address or register (0) - (12).

Parameters

The parameters are explained under the standard form of the ETDES macro with the following exception:

,MF=(E,*cntl addr)*
Specifies the execute form of the ETDES macro. This form uses a remote parameter list.

Chapter 11. ETDIS — Disconnect entry table

Description

The ETDIS macro disconnects one or more entry tables from the home address space's linkage table.

Related macros

ETDEF, ETCRE, ETCON, and ETDES

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in primary address space

Programming requirements

None.

Restrictions

An entry table, to be disconnected, must be connected to the home address space of the ETDIS issuer.

Input register information

The ETDIS macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter

- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the ETDIS macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.
- Otherwise, the caller must ensure that the following general purpose register contains the specified information:

  Register
  Contents

  13
  The address of an 18-word save area

Output register information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Performance implications

None.

Syntax

The ETDIS macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ETDIS.
ETDIS	
ƀ	One or more blanks must follow ETDIS.
TKLIST=*addr*	*addr*: RX-type address or register (0) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

TKLIST=*addr*
Specifies the address of a list of 1 to 32 fullword tokens, returned by the ETCRE macro, identifying the entry tables to be disconnected from the home address space's linkage table. The first entry of the list must be a fullword count of the number of tokens (1 to 32) in the list.

,RELATED=*value*
Specifies information used to self-document macros by "relating" functions or services to corresponding services performed elsewhere. The format and contents of the information specified can be any valid coding values.

ABEND codes

052
053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

When ETDIS macro returns control to your program, GPR 15 contains a return code.

Table 17. Return Code for the ETDIS Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: The entry table is successfully disconnected.

Examples

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in *z/OS MVS Programming: Extended Addressability Guide*.

Chapter 12. EVENTS — Wait for one or more events to complete

Description

The EVENTS macro is a functional specialization of the WAIT macro with the ECBLIST parameter, with the advantages of notifying the program that events have completed and the order in which they completed.

The macro performs the following functions:

- Creates and deletes EVENTS tables.
- Initializes and maintains a list of completed event control blocks.
- Provides for single or multiple ECB processing.

For a detailed explanation of how to use EVENTS to perform these functions, see "Using the EVENTS macro" in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

The description of the EVENTS macro follows. The EVENTS macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP* with the exception of the BRANCH=YES parameter.

Note: LOCAL lock means the local lock of the home address space.

Environment

The requirements for the caller are different for BRANCH=NO and BRANCH=YES.

If you specify BRANCH=NO, the requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

If you specify BRANCH=YES, the requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	Local lock must be held

Environmental factor	Requirement
Control parameters:	Must be in the primary address space

Programming requirements

If you specify BRANCH=YES, you must include the CVT mapping macro.

Restrictions

None.

Input register information

Before issuing the EVENTS macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The EVENTS macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede EVENTS.

Syntax	Description
EVENTS	
␣	One or more blanks must follow EVENTS.
ENTRIES=*n*	*n*: Decimal digits 1-32767
ENTRIES=*addr*	*addr*: Register (2) - (12).
ENTRIES=DEL,TABLE=*tab addr*	*tab addr*: Symbol, RX-type address, or register (2) - (12).
TABLE=*tab addr*	**Note**: If the ENTRIES parameter is specified as indicated in the first two formats, no other parameters may be specified.
,ECB=*ecb addr*	*ecb addr*: Symbol, RX-type address, or register (2) - (12).
,LAST=*last addr*	*last addr*: Symbol, RX-type address, or register (2) - (12).
	Note: If LAST is specified, WAIT must also be specified.
,WAIT=YES	**Note**: Do not specify WAIT=YES when running in a disabled state.
,WAIT=NO	
,BRANCH=NO	**Default**: BRANCH=NO
,BRANCH=YES	

Parameters

The parameters are explained below:

ENTRIES=*n*
ENTRIES=*addr*
Specifies either a register or a decimal number from 1 to 32,767 which specifies the maximum number of completed ECB addresses that can be processed in an EVENTS table concurrently.

Note: When this parameter is specified, no other parameter should be specified.

ENTRIES=DEL,TABLE=*tab addr*
Specifies that the EVENTS table whose address is specified by TABLE=*tab addr* is to be deleted. The user is responsible for deleting all of the tables he creates; however, all existing tables are automatically freed at task termination.

Note:

1. When this parameter is specified, no other parameter should be specified.
2. TABLE resides in 24-bit addressable storage.

TABLE=*tab addr*
Specifies either a register number or the address of a word containing the address of the EVENTS table associated with the request. The address specified with the operand TABLE must be that of an EVENTS table created by this task.

Note: TABLE resides in 24-bit addressable storage.

,WAIT=NO
,WAIT=YES
Specifies whether or not to put the issuing program in a wait state when there are no completed events in the EVENTS table (specified by the TABLE parameter).

,ECB=*ecb addr*
Specifies either a register number or the address of a word containing the address of an event control block. The EVENTS macro should be used to initialize any event-type ECB. To avoid the accidental destruction of bit settings by a system service such as an access method, the ECB should be initialized after the system service that will post the ECB has been initiated (thus making the ECB eligible for posting) and before the EVENTS macro is issued to wait on the EVENTS table.

Note:

1. Register 1 should not be specified for the ECB address.
2. This parameter may not be specified with the LAST parameter.
3. The ECB can reside above or below 16 megabytes.
4. If only ECB initialization is being requested, neither WAIT=NO nor WAIT=YES should be specified, to prevent any unnecessary WAIT processing from occurring.

,LAST=*last addr*
Specifies either a register number or the address of a word containing the address of the last EVENT parameter list entry processed.

Note:

1. Do not specify Register 1 for the LAST address.
2. Do not specify this parameter with the ECB parameter.
3. The WAIT macro must also be specified.
4. LAST resides in 24-bit addressable storage.

,BRANCH=NO
,BRANCH=YES
Specifies that an SVC entry (BRANCH=NO) or a branch entry (BRANCH=YES) is to be performed.

ABEND codes

The caller might encounter one of the following ABEND codes:

- 17A
- 17D
- 37A
- 37D
- 47A
- 47D
- 57D
- 67D
- 77D
- 87D

See *z/OS MVS System Codes* for explanations and responses for these codes.

Return and reason codes

None.

Example 1

The following shows total processing through EVENTS.

EVENTS and ECB Initialization

```
     EVENTS     ENTRIES=1000

     ST         R1,TABADD

     WRITE      ECBA

     LA         R2,ECBA...

     EVENTS     TABLE=TABADD,ECB=(R2)
```

Parameter List Processing

```
        EVENTS     TABLE=TABADD,WAIT=YES

        LR         R3,R1       PARMLIST ADDR
        B          LOOP2       GO TO PROCESS ECB
LOOP1   EVENTS     TABLE=TABADD,WAIT=YES,LAST=(R3)
        LR         R3,R1       SAVE POINTER
LOOP2   EQU        *           PROCESS COMPLETED EVENTS

        TM         0(R3),X'80' TEST FOR MORE EVENTS
        BO         LOOP1       IF NONE, GO WAIT
        LA         R3,4(,R3)   GET NEXT ENTRY
        B          LOOP2       GO PROCESS NEXT ENTRY
```

Deleting EVENTS Table

```
        EVENTS     TABLE=TABADD,ENTRIES=DEL

TABADD  DS         F
```

Example 2

Processing One ECB at a Time.

```
          EVENTS ENTRIES=10            CREATE EVENTS TABLE
          ST    R1,TABLE               SAVE EVENTS TABLE
 *                                     ADDRESS
 NEXTREC  GET   TPDATA,KEY             GET KEY OF NEXT RECORD
 *                                     TO PROCESS
          READ  DECBRW,KU,,'S',MF=E     READ THE RECORD
          LA    R3,DECBRW               POINT TO ECB
          EVENTS TABLE=TABLE,ECB=(R3),WAIT=YES  ADD ECB TO
 *                                     TABLE AND WAIT UNTIL
 *                                     IT IS POSTED
 *        PROCESS THE RECORD
          WRITE DECBRW,K,MF=E           WRITE OUT THE RECORD
          LA    R3,DECBRW               POINT TO THE ECB
          EVENTS TABLE=TABLE,ECB=(R3),WAIT=NO
          B     CKRETEST                GO SEE IF IT'S POSTED
 RETEST   EVENTS TABLE=TABLE,WAIT=NO    CHECK TO SEE IF ECB IS
 *                                     POSTED
 CKRETEST LTR   R1,R1                  ANY ECBS POSTED?
          BNZ   NEXTREC                BRANCH IF YES - NEXT
 *                                     RECORD
          B     RETEST                 ELSE KEEP CHECKING
 TABLE    DS    A                      ADDRESS OF EVENTS TABLE
```

Chapter 13. EXTRACT — Extract TCB information

Description

The EXTRACT macro causes the system to provide information from specified fields of the task control block or a subsidiary control block for either the active task or one of its subtasks. The system places the information in an area that the program provides. For a description of this area see “Providing an EXTRACT Answer Area” in *z/OS MVS Programming: Authorized Assembler Services Guide*. When EXTRACT is issued, its parameter list can reside in 24 or 31-bit addressable storage.

To obtain the address of a TIOT entry, you can use either the GETDSAB macro or the EXTRACT macro.

Your installation might have installed products that require the use of the GETDSAB macro to obtain the address of the products' TIOT entries. If you plan to use the EXTRACT macro, first check the documentation for the related product to ensure that the product does not require the use of the GETDSAB macro.

Note:

1. For procedures for using GETDSAB to obtain the address of a TIOT entry and the UCB address, see *z/OS MVS Programming: Authorized Assembler Services Guide*.
2. If the EXTRACT macro is used to obtain the TIOT in order to find the UCB, it is the user's responsibility to ensure that the TIOT contains the UCB address. For procedures for finding the UCB address, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, and user key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Performance implications

None.

Syntax

The standard form of the EXTRACT macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EXTRACT.
EXTRACT	
ƀ	One or more blanks must follow EXTRACT.
answer addr	*answer addr*: A-type address, or register (2) - (12).
,'S'	**Default**: 'S'
,tcb addr	*tcb addr*: A-type address, or register (2) - (12).
,FIELDS=*(tcb info)*	*tcb info*: Any combination of the following, separated by commas: **ALL** PRI **GRS** CMC **FRS** TIOT **AETX** COMM **TSO** PSB **TJID** ASID

Parameters

The parameters are explained as follows:

answer addr
Specifies the address of the answer area to contain the requested information. The area is one or more fullwords, starting on a fullword boundary. The number of fullwords must be the same as the number of fields specified in the FIELDS parameter, unless ALL is coded. If ALL is coded, seven fullwords are required.

,'S'
,tcb addr
Specifies the address of a fullword on a fullword boundary containing the address of a task control block for a subtask of the active task. If 'S' is coded or is the default, no address is specified and the active task is assumed.

,FIELDS=*(tcb info)*
Specifies the task control block information requested:

ALL
Requests information from the GRS, FRS, reserved, AETX, PRI, CMC, and TIOT fields. (If ALL is specified, 7 words are required just for ALL.)

GRS
Is the address of the save area used by the system to save the general purpose registers 0-15 when the task is not active.

FRS
Is the address of the save area used by the system to save the floating point registers 0, 2, 4, and 6 when the task is not active.

AETX
Is the address of the end-of-task exit routine specified in the ETXR parameter of the ATTACH (or ATTACHX) macro used to create the task.

PRI
Is the current limit (third byte) and dispatching (fourth byte) priorities of the task. The two high-order bytes are set to zero.

CMC
Is the task completion code. If the task is not complete, the field is set to zero.

TIOT
Is the address of the task input/output table.

COMM
Is the address of the command scheduler communications list. The list consists of a pointer to the communications event control block and a pointer to the command input buffer, and a token. (If a token exists, the high-order bit of the token field is set to one). The token is used only with internal START commands. See "Issuing an Internal START or REPLY Command" in *z/OS MVS Programming: Authorized Assembler Services Guide*.

TSO
Is the address of a byte in which a high-order bit of 1 indicates a TSO/E address space initiated from the LOGON command (that is, in a foreground TSO/E session). A high-order bit of 0 indicates either background TSO/E or a non-TSO/E address space.

PSB
Is the address of the TSO/E protected step control block and is returned:

- In a foreground TSO/E session (initiated through LOGON)
- In a background TSO/E session (initiated through the TSO/E TMP, IKJEFT01).
- In a TSO/E environment initialized outside of the TSO/E TMP (initiated through the IKJTSOEV service).

TJID
Is the address space identifier (ASID) for a foreground TSO/E session (initiated through LOGON), or zero for either background TSO/E or a non-TSO/E address space.

ASID
Is the address space identifier.

ABEND codes

The EXTRACT macro might abnormally terminate with one of the following abend codes: X'128', X'228', and X'328'. See *z/OS MVS System Codes* for explanations and programmer responses.

Return and reason codes

None.

Example 1

Provide information from all the fields of the indicated TCB except ASID. WHERE is the label of the answer area, ADDRESS is the label of a fullword that contains the address of the subtask TCB for which information is to be extracted.

```
EXTRACT WHERE,ADDRESS,FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

Example 2

Provide information from the current TCB, as above.

```
EXTRACT WHERE,'S',FIELDS=(ALL,TSO,COMM,PSB,TJID)
```

Example 3

Provide information from the command scheduler communications list. ANSWER is the label of the answer area and TCBADDR is the label of a fullword that contains the address of the subtask TCB from which information is to be extracted.

```
EXTRACT ANSWER,TCBADDR,FIELDS=(COMM)
```

EXTRACT - List form

The list form of the EXTRACT macro is used to construct a remote control program parameter list.

Syntax

The list form of the EXTRACT macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EXTRACT.
EXTRACT	
ƀ	One or more blanks must follow EXTRACT.
answer addr	*answer addr*: A-type address.
,'S'	**Default**: 'S'
,tcb addr	*tcb addr*: A-type address.

Syntax	Description
,FIELDS=*(tcb info)*	*tcb info*: any combination of the following, separated by commas: **ALL** PRI **GRS** CMC **FRS** TIOT **AETX** COMM **TSO** PSB **TJID** ASID
,MF=L	

Parameters

The parameters are explained under the standard form of the EXTRACT macro, with the following exception:

,MF=L
Specifies the list form of the EXTRACT macro.

EXTRACT - Execute form

The execute form of the EXTRACT macro uses, and can modify, a remote control program parameter list. If the FIELDS parameter, restricted in use, is coded in the execute form, any TCB information specified in a previous FIELDS parameter is canceled and must be respecified if required for this execution of the macro.

Syntax

The execute form of the EXTRACT macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EXTRACT.
EXTRACT	
ƀ	One or more blanks must follow EXTRACT.

Syntax	Description
answer addr	*answer addr*: RX-type address, or register (2) - (12).
,'S'	*tcb addr*: RX-type address, or register (2) - (12).
,tcb addr	
,FIELDS=*(tcb info)*	*tcb info*: any combination of the following, separated by commas: **ALL** PRI **GRS** CMC **FRS** TIOT **AETX** COMM **TSO** PSB **TJID** ASID
,MF=(E,*ctrl addr)*	*ctrl addr: RX-type address, or register (1) or (2) - (12).*

Parameters

The parameters are explained under the standard form of the EXTRACT macro, with the following exception:

,MF=(E,*ctrl addr)*
Specifies the execute form of the EXTRACT macro using a remote control program parameter list.

Chapter 14. FESTAE — Fast extended STAE

Description

The FESTAE macro allows an SVC to define and activate, or to deactivate and no longer define, an ESTAE-type recovery routine with minimal overhead and no locking requirements. The ESTAE-type recovery routine activated by FESTAE receives control in the same sequence and under the same conditions as it would if it were activated by the ESTAE macro. The FESTAE macro can be issued in cross memory mode as long as the currently addressable address space is the home address space. For more information, see *z/OS MVS Programming: Authorized Assembler Services Guide*. To delete a FESTAE recovery routine that was established by the FESTAE macro, use the FESTAE macro rather than macros such as ESTAE, ESTAEX, or STAE.

The FESTAE macro expansion has no external linkage.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in the primary address space Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

Programming requirements

FESTAE users executing in 31-bit addressing mode must recompile using the FESTAE macro expansion so that the exit routine gets control in 31-bit addressing mode.

The caller must include the following mapping macros:

- IHAPSA
- IHARB
- IHASCB
- IKJTCB

Restrictions

- Only type 2, 3, or 4 SVC routines can use the FESTAE macro
- The FESTAE macro can be issued to create only one recovery routine within the scope of the SVC routine. The ESTAEX macro or the ESTAE macro with the BRANCH option must be used to create additional recovery routines.

Input register information

Before issuing the FESTAE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register. Register notation is required for the following FESTAE macro parameters: EXITADR, WRKREG, RBADDR, TCBADDR, and PARAM.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Unchanged

1-14
One of the following:

- If you specify 0,WRKREG=*work reg addr*, the register you specify (1-14) is used as a work register by the system.
- If you specify EXITADR=*exit addr*, the register you specify (1-14) is used as a work register by the system.
- Registers not specified for either *work reg addr* or *exit addr* are unchanged.

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

Specification of the TCBADDR keyword results in more efficient code.

Syntax

The FESTAE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FESTAE.
FESTAE	

Syntax	Description
ƀ	One or more blanks must follow FESTAE.
EXITADR=*exit addr*	*exit addr*: Register (1) - (14).
0,WRKREG=*work reg*	*work reg addr*: Register (1) - (14).
,RBADDR=*svrb addr*	*svrb addr*: Register (1) - (14).
,TCBADDR=*tcb addr*	*tcb addr*: Register (1) - (14).
,PARAM=*list addr*	*list addr*: Register (1) - (14).
,XCTL=NO	**Default**: XCTL=NO
,XCTL=YES	
,PURGE=NONE	**Default**: PURGE=NONE
,PURGE=HALT	
,PURGE=QUIESCE	
,ASYNCH=YES	**Default**: ASYNCH=YES
,ASYNCH=NO	
,TERM=NO	**Default**: TERM=NO
,TERM=YES	
,RECORD=NO	**Default**: RECORD=NO
,RECORD=YES	
,ERRET=*label*	*label*: Any valid assembler name.
,SDWALOC31=NO	**Default**: SDWALOC31=NO
,SDWALOC31=YES	

Parameters

The parameters are explained as follows:

EXITADR=*exit addr*
0,WRKREG=*work reg*
Specifies whether an ESTAE-type recovery routine is to be defined and activated, or deactivated and no longer defined. EXITADR=*exit addr* specifies the register that contains the address of an ESTAE-type recovery routine to be entered if the task issuing FESTAE ends abnormally.

If you specify 0,WRKREG=*work reg*, the current ESTAE-type recovery routine is deactivated and no longer defined if it was defined by the FESTAE macro. An error occurs if the current ESTAE-type recovery routine was not created by FESTAE. You do not have to initialize the register you specify for *work reg*; the system uses it as a work register.

,RBADDR=*svrb addr*
Specifies a register that contains the address of the current SVRB prefix. RBADDR must be specified if EXITADR has also been specified.

,TCBADDR=*tcb addr*
Specifies the register containing the current TCB address.

,PARAM=*list addr*
Specifies the register containing the address of a user-defined parameter list that contains data to be used by the ESTAE routine. The routine receives this address when it is scheduled for execution. The use of this parameter list is optional, but the user should zero out any spurious data it might contain whether or not he intends to use it. If the user does not select the PARAM option, the routine receives instead the 24-byte parameter area in the SVRB. In this case, the user must locate this SVRB parameter area and initialize it with appropriate data.

,ERRET=*label*
Specifies a label within the CSECT issuing the FESTAE for which addressability has been established. The FESTAE macro branches to this label if it is returning a code other than zero. This option saves the user the instructions necessary to check the return code. If the user does not specify the ERRET option, control returns instead to the instruction immediately following the FESTAE macro. The return code is in register 15.

All the other FESTAE parameters have the same meaning as their ESTAE counterparts.

ABEND codes

None.

Return codes

When control is returned to the instruction following the FESTAE macro, GPR 15 contains one of the following return codes.

Table 18. Return Codes for the FESTAE Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Successful completion of the FESTAE request. **Action** None.
08	**Meaning**: Program error. A previous create has been issued with FESTAE for this SVRB; the request has been ignored. **Action**: None; do not reissue this macro.
0C	**Meaning**: Program error. Cancel has been specified under one of the following conditions: • There is no exit for this TCB. • The most recent exit is not owned by the caller. • The most recent exit was not created by FESTAE. **Action**: Ensure that the current recovery routine was established using the FESTAE macro.

Example

In case of an abnormal termination, execute the ESTAE routine specified by register 2, allow asynchronous processing, do not allow special error processing, default to PURGE=NONE, and pass the parameter list pointed to by register 7 to the ESTAE routine.

```
FESTAE  EXITADR=(REG2),RBADDR=(REG3),TCBADDR=(REG6),   X
        PARAM=(REG7),ASYNCH=YES,TERM=NO
```

Chapter 15. FRACHECK — Check user's authorization (for RACF Release 1.8.1 or earlier)

See *z/OS Security Server RACROUTE Macro Reference* for a description of this macro.

Chapter 16. FREEMAIN — Free virtual storage

Description

Use the FREEMAIN macro to free one or more areas of virtual storage. You can also use the FREEMAIN macro to free an entire virtual storage subpool if it is owned by the task under which your program is issuing the FREEMAIN. For more information on releasing a subpool, see the chapter about virtual storage management in *z/OS MVS Programming: Assembler Services Guide*.

You can also use the STORAGE macro to free storage, even if the storage was obtained using the GETMAIN macro. Compared to FREEMAIN, STORAGE provides an easier-to-use interface and has no restrictions or locking requirements. See the chapter about virtual storage management in *z/OS MVS Programming: Authorized Assembler Services Guide* for a comparison of FREEMAIN and STORAGE.

The FREEMAIN macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the BRANCH parameter.

The FREEMAIN macro provides two types of entry linkage: SVC entry and branch entry. If you do not specify the BRANCH parameter, the FREEMAIN service receives control through **SVC entry**. If you specify the BRANCH parameter, the FREEMAIN service receives control through **branch entry**.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For subpools 0-127: problem state and PSW key 8-15. For subpools 131 and 132, **one or more of the following**: • Supervisor state • PSW key 0-7 • APF-authorization. • PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be released. For other subpools, **one or more of the following**: • Supervisor state • PSW key 0-7 • APF-authorized. To issue a subpool release for subpool 0: PSW key 0. For branch entry: supervisor state and PSW key 0.
Dispatchable unit mode:	For SVC entry: task. For branch entry: task or SRB.
Cross memory mode:	For SVC entry: PASN=HASN=SASN. For branch entry: any PASN, any HASN, any SASN.

Environmental factor	Requirement
AMODE:	For SVC entry: 24- or 31- or 64-bit. For branch entry: 24- or 31-bit. • For RU, RC requests: The system treats all addresses and values as 31-bit. • For all other requests: If the calling program is in 31-bit mode, the system treats all addresses and values, passed to the FREEMAIN macro, as 31-bit. Otherwise, the system treats addresses and values as 24-bit.
RMODE:	For SVC-entry, includes 64-bit
ASC mode:	For BRANCH=(YES,GLOBAL), primary or access register (AR). For all other requests, primary. Callers in AR mode must use BRANCH=(YES,GLOBAL) and can obtain only global (common) storage.
Interrupt status:	For BRANCH=(YES,GLOBAL), disabled for I/O and external interrupts. For all other requests, enabled for I/O and external interrupts.
Locks:	• For SVC entry, no locks may be held. • For BRANCH=YES, your program must hold the local lock for the currently addressable address space. • For BRANCH=YES, when running in cross-memory mode, your program must hold the CML lock for the currently addressable address space. • For BRANCH=(YES,GLOBAL), your program must be in an MVS-recognized state of disablement, which can be achieved by obtaining the CPU lock.
Control parameters:	For LC, LU, L, VC, VU, V, EC, EU, E requests: control parameters must be in the primary address space. For other requests: control parameters are in registers.

Programming requirements

Before issuing the FREEMAIN macro in AR mode, issue SYSSTATE ASCENV=AR.

Restrictions

- Parameters passed to the FREEMAIN macro must not reside within the area being freed. If this restriction is violated and the parameters are the last allocated areas on a virtual page, the whole page is freed and FREEMAIN ends abnormally with an X'0C4' abend code.
- The current task ends abnormally if the specified virtual storage area does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not owned by the task identified as the owner of the storage.
- For SVC entry, the caller cannot have an EUT FRR established.

Input register information for SVC entry

Before issuing the FREEMAIN macro without the BRANCH parameter (SVC entry), the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information for SVC entry

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
For a conditional request, contains the return code. For an unconditional request, used as a work register by the system.

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Input register information for BRANCH=YES

Before issuing the FREEMAIN macro with BRANCH=YES, the caller must ensure that the following GPRs contain the specified information:

Register
Contents

4
The address of the input TCB, if you are releasing private storage.

Set GPR 4 to 0 or the address of a TCB in the currently addressable address space. Setting GPR 4 to 0 identifies the input TCB as the TCB that owns the cross-memory resources for the currently addressable address space (task whose TCB address is in ASCBXTCB).

For an explanation of the term **input TCB**, and to determine system-assigned defaults for private storage ownership, see the topic about selecting the right subpool for virtual storage requests in *z/OS MVS Programming: Authorized Assembler Services Guide*.

7
The address of the ASCB for the currently addressable address space.

Output register information for BRANCH=YES

For RC, RU, VRC, and VRU requests: when control returns to the caller, GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2
Unchanged.

3
Used as a work register by the system.

4-13
Unchanged.

14
Used as a work register by the system.

15
For a conditional request, contains the return code. For an unconditional request, used as a work register by the system.

For all other requests: when control returns to the caller, GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
For a conditional request, contains the return code. For an unconditional request, used as a work register by the system.

When control returns to the caller, ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

Input register information for BRANCH=(YES,GLOBAL)

Before issuing the FREEMAIN macro with BRANCH=(YES,GLOBAL), you are not required to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information for BRANCH=(YES,GLOBAL)

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2
Unchanged.

3-4
Used as work registers by the system.

5-13
Unchanged.

14
Used as a work register by the system.

15
For a conditional request, contains the return code. For an unconditional request, used as a work register by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the service returns control.

Performance implications

None.

Syntax

The standard form of the FREEMAIN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FREEMAIN.
FREEMAIN	
ƀ	One or more blanks must follow FREEMAIN.
LC,LA=*length addr*	*length addr*: A-type address, or register (2) - (12).
LU,LA=*length addr*	
L,LA=*length addr*	
VC	
VU	
V	
EC,LV=*length value*	*length value*: symbol, decimal number, or register (2) - (12).
EU,LV=*length value*	

Syntax	Description
E,LV=*length value*	
RC,LV=*length value*	If R, RC, or RU is specified, register (0) may also be used.
RC,SP=*subpool nmbr*	*subpool nmbr*: symbol, decimal number 0-255, or register (2) - (12). If R is specified, register (0) may also be used. **Note:** For a subpool release (RC,SP or RU,SP, or R,SP), no other parameters except RELATED and BRANCH=YES can be specified.
RU,LV=*length value*	
RU,SP=*subpool nmbr*	
R,LV=*length value*	
R,SP=*subpool nmbr*	
,A=*addr*	*addr*: A-type address, or register (2) - (12). If R, RC, or RU is specified, register (1) can also be used.
	Note: If R, RC, or RU is specified, register (1) can also be specified.
,SP=*subpool nmbr*	*subpool nmbr*: symbol, decimal number 0-255, or register (2) - (12).
	Default: SP=0. If R is specified, register (0) may also be used.
,BRANCH=YES	**Note:** BRANCH=(YES,GLOBAL) may be specified only with RC or RU.
,BRANCH=(YES,GLOBAL)	
,KEY=*number*	*nmbr*: decimal numbers 0-15, or register (2) - (12).
	Note: KEY may be specified only with RC or RU.
,RELATED=*value*	*value*: any valid assembler character string.

Parameters

The parameters are explained as follows:

LC,LA=*length addr*
LU,LA=*length addr*
L,LA=*length addr*
VC
VU
V
EC,LV=*length value*
EU,LV=*length value*
E,LV=*length value*
RC,LV=*length value*
RC,SP=*subpool nmbr*
RU,LV=*length value*
RU,SP=*subpool nmbr*
R,LV=*length value*
R,SP=*subpool nmbr*

Specifies the type of FREEMAIN request:

LC, LU, and L indicate conditional (LC) and unconditional (LU and L) list requests and specify release of one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas must be provided in a corresponding list whose address is specified in the A parameter. All virtual storage areas must start on a doubleword boundary.

VC, VU, and V indicate conditional (VC) and unconditional (VU and V) variable requests and specify release of single areas of virtual storage. The address and length of the virtual storage area are provided at the address specified in the A parameter.

EC, EU, and E indicate conditional (EC) and unconditional (EU and E) element requests and specify release of single areas of virtual storage. The length of the single virtual storage area is indicated in the LV parameter. The address of the virtual storage area is provided at the address indicated in the A parameter.

RC, RU, and R indicate conditional (RC) and unconditional (RU and R) register requests and specify either the release of all the storage in a subpool or the release of a certain area in a subpool. For information on how to release all the storage in a subpool, see the description for the SP parameter. If the release is for a certain area in a subpool, the address of the virtual storage area is indicated in the A parameter. The length of the area is indicated in the LV parameter. The virtual storage area must start on a doubleword boundary.

Note:

1. For a conditional request, errors detected while processing a FREEMAIN request with incorrect or inconsistent parameters cause the FREEMAIN service to return to the caller with a non-zero return code. For all other errors, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.

   For an unconditional request, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.
2. If the address of the area to be freed is above 16 megabytes, you must use RC or RU.

LA specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. One word is required for each virtual storage area to be released; the high-order bit in the last word must be set to 1 to indicate the end of the list. Each word must contain the required length in the low-order three bytes. The fullwords in this list must correspond with the fullwords in the associated list specified in the A parameter. The words must not be in the area to be released. If this rule is violated and if the words are the last allocated items on a virtual page, the whole page is returned to storage and the FREEMAIN abends with an X'0C4' abend code.

LV specifies the length, in bytes, of the virtual storage area being released. The value should be a multiple of 8; if it is not, the control program uses the next high multiple of 8.

- If you specify R,LV=(0) you cannot specify the SP parameter. You must specify the subpool in register 0; the high-order byte must contain the subpool number and the low-order three bytes must contain the length unless you are requesting a subpool release. On a subpool release, the low-order three bytes must contain zeros.
- If you specify R,LV using a symbol, decimal number, or register 2-12, you can specify the SP parameter using registers 0 or 2-12.

,A=*addr*
Specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary.

- If E, EC, or EU is coded, one word is required, which contains the address of the virtual storage area to be released.
- If V, VC, or VU is coded, two words are required; the first word contains the address of the virtual storage area to be released, and the second word contains the length of the area to be released.
- If L, LC, or LU is coded, one word is required for each virtual storage area to be released; each word contains the address of one virtual storage area.
- If R, RC, or RU is coded, one word is required, which contains the address of the virtual storage area to be released. If R, RC, or RU is coded and *addr* specifies a register, register 1 through 12 can be used and must contain the address of the virtual storage area to be released.

Do not specify a storage address of 0 with a storage length of 0. This combination causes FREEMAIN to free the subpool specified with the SP parameter, or subpool 0 if the SP parameter is omitted.

,SP=*subpool nmbr*
Specifies the subpool number of the virtual area to be released. Valid subpools numbers are between 0 and 255. The SP parameter is optional and if omitted, subpool 0 is assumed. If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero.

A request to release all the storage in a subpool is known as a **subpool release**. To issue a subpool release, specify RC,SP or RU,SP or R,SP, and do not use the A or the KEY parameter. The following subpools are valid on the SP parameter for a subpool release: 0-127, 129-132, 203-204, 213-214, 223-224, 229-230, 233, 236-237, 240, 249, and 250-253. An attempt to issue a subpool release for any other subpool causes an abend X'478' or X'40A'. For information about subpools, see *z/OS MVS Programming: Assembler Services Guide* and *z/OS MVS Programming: Authorized Assembler Services Guide*.

Note:

1. Callers executing in supervisor state and PSW key 0, who specify subpool 0, will free storage from subpool 252. Therefore, when requesting a dump of this storage through the SDUMP macro, the caller must specify subpool 252 rather than subpool 0.
2. Requests for storage from subpools 240 and 250 are translated to subpool 0 storage requests.

,BRANCH=YES
,BRANCH=(YES,GLOBAL)
Specifies that a branch entry is to be used.

BRANCH=YES allows both local (private area) and global (common area) storage to be released. See Input register information for BRANCH=YES for specific information on input register requirements.

BRANCH=(YES,GLOBAL) allows only global storage to be released. With BRANCH=(YES,GLOBAL), the SP parameter may designate only subpools 226-228, 231, 239, 241, 245, 247, or 248. BRANCH=(YES,GLOBAL) is valid only with RC or RU.

,KEY=*key number*
Specifies the storage key in which the storage was obtained. The valid storage keys are 0-15. If a register is specified, the storage key must be in bits 24-27 of the register. KEY can be specified for the following subpools: 129-132, 227-231, 241, and 249. BRANCH is required with KEY for subpools 227-231, 241, and 249. BRANCH=(YES,GLOBAL) is not valid for subpools 129-132, 229-230, and 249.

,RELATED=*value*
Specifies information used to self-document macros by "relating" functions or services to corresponding functions or services and can be any valid assembler character string.

ABEND codes

Abend codes FREEMAIN might issue are listed below in hexadecimal. For detailed abend code information, see *z/OS MVS System Codes*.

- 105
- 10A
- 178
- 205
- 20A
- 278
- 305
- 30A
- 378
- 40A
- 478
- 505
- 605
- 705
- 70A
- 778
- 805
- 80A
- 878
- 905
- 90A
- 978
- A05
- A0A
- A78
- B05
- B0A
- B78
- D05
- D0A
- D78

Return and reason codes

When the FREEMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

Table 19. Return Codes for the FREEMAIN Macro

Return Code	Meaning and Action
0	**Meaning**: Successful completion. **Action**: None.
4	**Meaning**: Program error. Not all requested virtual storage was freed. **Action**: Check your program for the following kinds of errors: • The address of the storage area to be freed is not correct. • The subpool you have specified does not match the subpool of the storage to be freed. • The key you have specified does not match the key of the storage to be freed. • For private storage: the owning task identified by the input TCB is not correct for the storage to be freed.
8	**Meaning**: Program error. No virtual storage was freed because part of the storage area to be freed is fixed. **Action**: Determine whether you have made one of the following errors. If so, correct your program and rerun it: • You passed an incorrect storage area address to the FREEMAIN macro. • You attempted to free storage that is fixed.

Example 1

Free 400 bytes of storage from subpool 10. Register 1 contains the address of the storage area. If the storage is not allocated to the current task, do not abnormally terminate the caller.

```
FREEMAIN RC,LV=400,A=(1),SP=10
```

Example 2

Free all of subpool 3 (if any) that belongs to the current task. If the request is not successful, abnormally terminate the caller.

```
FREEMAIN RU,SP=3
```

Example 3

Free from subpool 5, three areas of storage of 200, 800, and 32 bytes, previously obtained using the list and execute forms of the GETMAIN macro. Storage area addresses are in AREAADD. If any of the storage areas to be freed are not allocated to the current task, abnormally terminate the caller.

```
FREEMAIN LU,LA=LNTHLIST,A=AREAADD,SP=5
  .
  .
  .
LNTHLIST DC F'200',F'800',X'80',FL3'32'
AREAADD  DS 3F
```

Example 4

Free 400 bytes of storage from default subpool 0 using branch entry. The address of the storage area is in register 2. If the request is not successful, do not abnormally terminate the caller.

```
FREEMAIN EC,LV=400,A=(2),BRANCH=YES
```

Example 5

Free 48 bytes of storage from subpool 231 using global branch entry. Register 5 contains the address of the storage area. Register 3 contains the storage key of the storage to be released. If the request is unsuccessful, abnormally terminate the caller.

```
FREEMAIN RU,LV=48,A=(5),SP=231,KEY=(3),BRANCH=(YES,GLOBAL)
```

FREEMAIN - List form

Use the list form of the FREEMAIN macro to construct a nonexecutable control program parameter list.

The list form of the FREEMAIN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FREEMAIN.
FREEMAIN	
ƀ	One or more blanks must follow FREEMAIN.
LC	
LU	
L	
VC	
VU	
V	
EC	
EU	
E	
,LA=*length addr*	*length addr*: A-type address.
,LV=*length value*	*length value*: symbol or decimal number.
	Note: 1. LA may only be specified with LC, LU, or L above. 2. LV may only be specified with EC, EU, or E above.
,A=*addr*	*addr*: A-type address.

Syntax	Description
,SP=*subpool nmbr*	*subpool nmbr*: symbol or decimal number.
,RELATED=*value*	*value*: any valid assembler character string.
,MF=L	

Parameters

The parameters are explained under the standard form of the FREEMAIN macro, with the following exceptions:

,MF=L
Specifies the list form of the FREEMAIN macro.

FREEMAIN - Execute form

A remote control program parameter list is used in, and can be modified by, the execute form of the FREEMAIN macro. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN.

The execute form of the FREEMAIN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FREEMAIN.
FREEMAIN	
ƀ	One or more blanks must follow FREEMAIN.
LC	
LU	
L	
VC	
VU	
V	
EC	
EU	
E	

Syntax	Description
,LA=*length addr*	*length addr*: RX-type address or register (2) - (12).
,LV=*length value*	*length value*: symbol, decimal number, or register (2) - (12).
	Note: 1. LA may only be specified with LC, LU, or L above. 2. LV may only be specified with EC, EU, or E above.
,A=*addr*	*addr*: RX-type address, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr*: symbol, decimal number, or register (0) or (2) - (12).
,BRANCH=YES	
,RELATED=*value*	*value*: any valid assembler character string.
,MF=(E,*list addr*)	*list addr*: RX-type address, or register (1) or (2) - (12).

Parameters

The parameters are explained under the standard form of the FREEMAIN macro, with the following exceptions:

,MF=(E,*list addr)*
Specifies the execute form of the FREEMAIN macro using a remote control program parameter list.

Chapter 17. FXECNTRL — Maintain Function Exploitation and Enablement

Description

The FXECNTRL macro allows fields in the IBM Function Registry for z/OS to be updated.

Additional references and an overview of the IBM Function Registry for z/OS can be found in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. PSW key 8-15 Only callers in supervisor state, key 0-7, or APF authorized are allowed to use: • REQUEST=APPLYIPLPARM • Value AUTHONLY for parameter FUNCTIONUPDTYPE • Value VALUE for parameter FUNCTIONUPDTYPE in combination with value 1 for parameter FUNCUPDTYPVALUE
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). Control parameters can be above 2G for an AMODE64 caller. The user-provided text strings referenced by the VENDORNAME, PRODUCTNAME, FUNCTIONNAME, PRODUCTID, and INSTANCEID have the same requirements and restrictions as the control parameters.

Programming Requirements

The caller can include the FXEZCTRL macro to get equate symbols for the return and reason codes.

Restrictions

None.

Input Register Information

Before issuing the FXECNTRL macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, when register 15 is not 0.

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

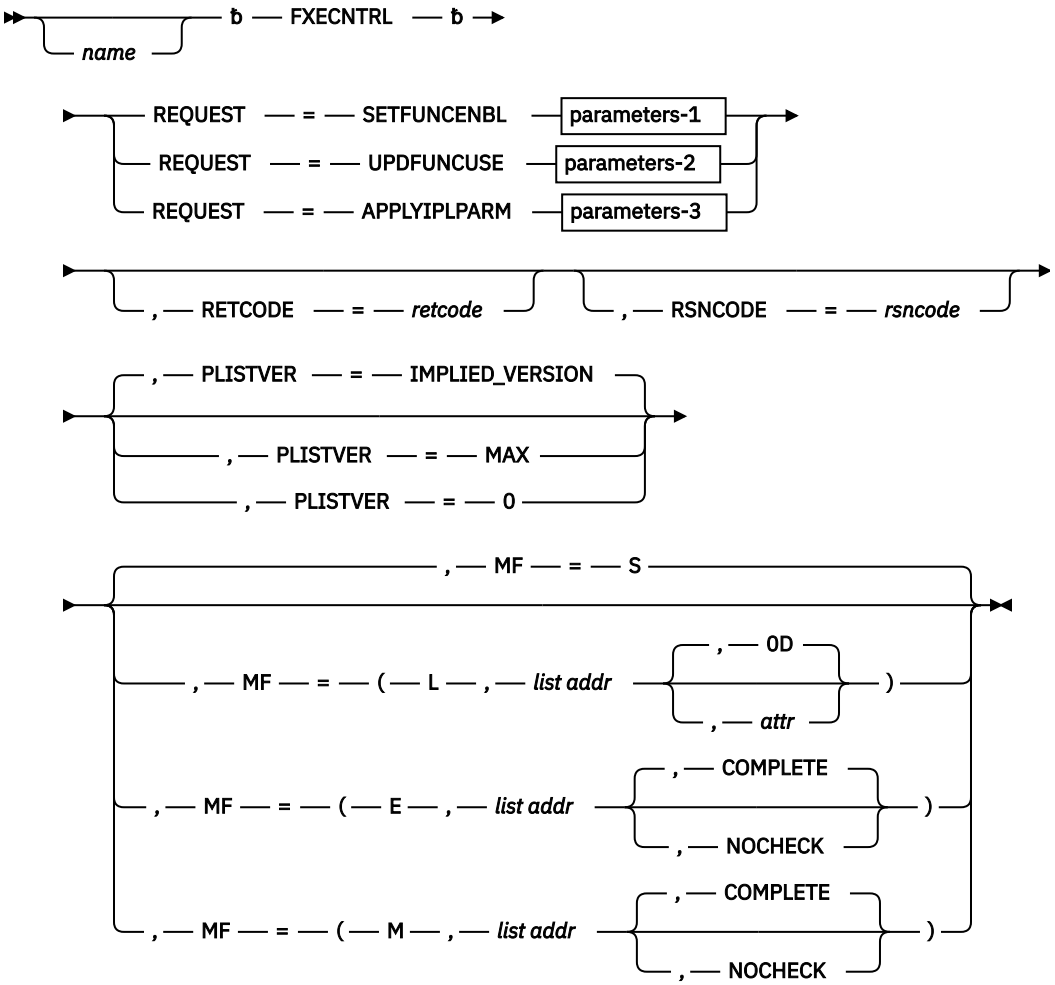
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

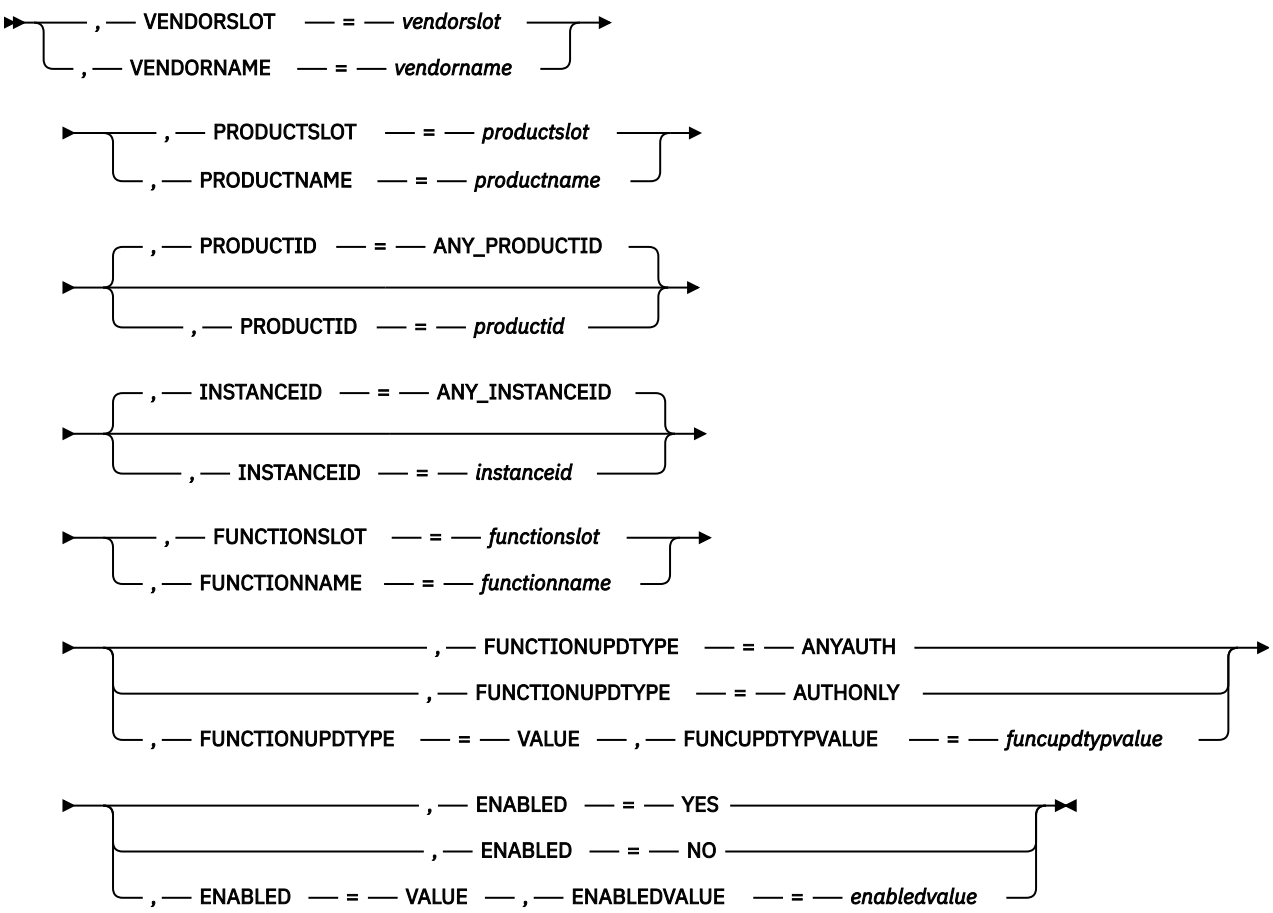
None.

Syntax

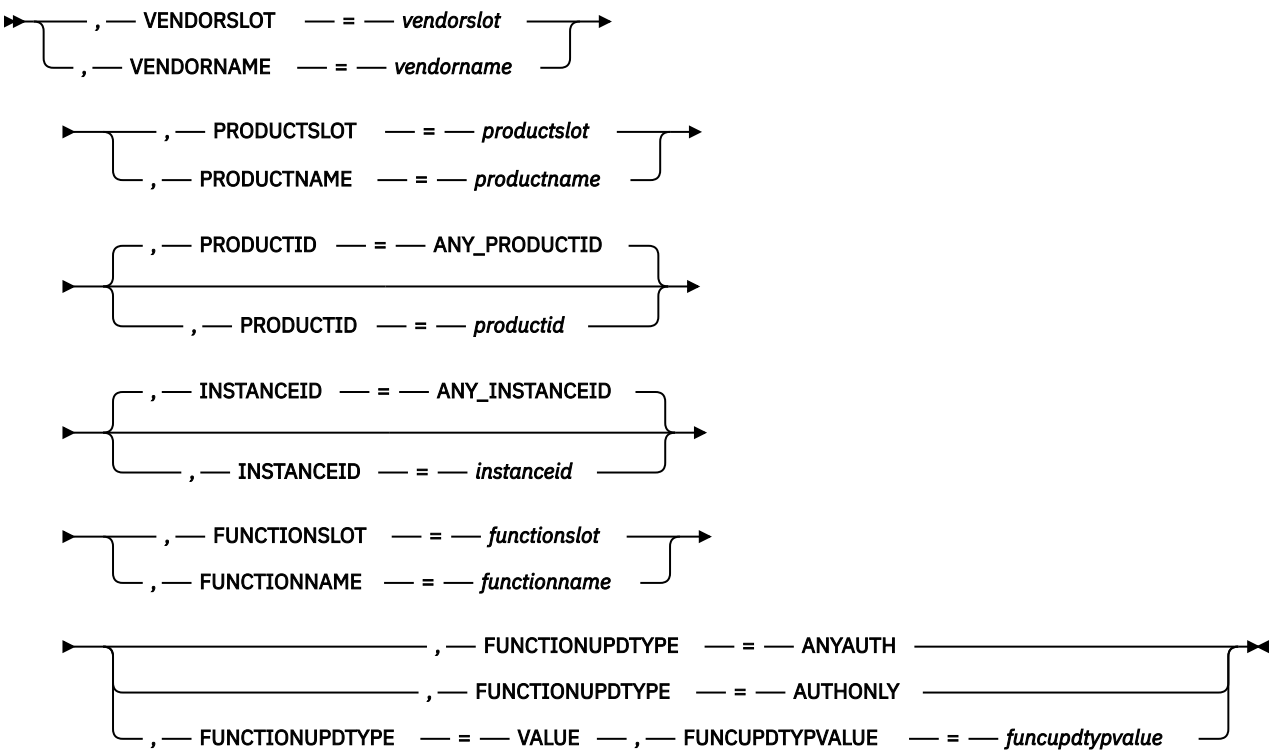
main diagram



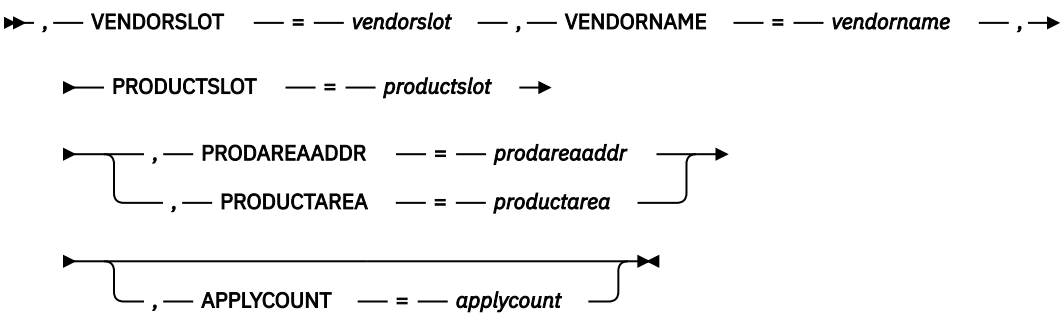
parameters-1



parameters-2



parameters-3



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the FXECNTRL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,APPLYCOUNT=*applycount*
When REQUEST=APPLYIPLPARM is specified, an optional output parameter, field that will contain the count of successful APPLY operations of deferred-apply statements to function entries in the registry for this APPLYIPLPARM request.

Note:

- This is not a count of how many unique function entries were affected by this request, since a single function entry could be a match for more than one statement and the overall APPLYCOUNT is incremented for each of those matching statements.
- A non-zero RETCODE and RSNCODE combination makes the information returned in this field incomplete (for example, unsuccessful APPLY operations for individual function entries are not counted) or even undefined (for example for parameter list access errors or service authorization errors).

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,ENABLED=YES
,ENABLED=NO
,ENABLED=VALUE
When REQUEST=SETFUNCENBL is specified, a required parameter, which indicates the value to set the selected function's enablement state to.

Note that some function entries might not be set up to have its enablement state changed or tracked at all. In such a case the request will end with a non-zero return code and the enablement state will not be changed.

,ENABLED=YES
indicates to change the enablement state to enabled.

,ENABLED=NO
indicates to change the enablement state to disabled.

,ENABLED=VALUE
indicates that the desired enablement state is specified as runtime value via parameter ENABLEDVALUE.

,ENABLEDVALUE=*enabledvalue*
When ENABLED=VALUE and REQUEST=SETFUNCENBL are specified, a required input parameter, which indicates the desired enablement state:

- specify 1 (one, see also equate *_XENABLED_YES) for enabled.
- specify 0 (zero, see also equate *_XENABLED_NO) for disabled.

To code: Specify the RS-type address of an 8 bit field.

,FUNCTIONNAME=*functionname*
When REQUEST=SETFUNCENBL is specified, a required input parameter which contains the name of the function that is to be used for the requested operation. If FUNCTIONNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,FUNCTIONNAME=*functionname*
When REQUEST=UPDFUNCUSE is specified, a required input parameter which contains the name of the function that is to be used for the requested operation. If FUNCTIONNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,FUNCTIONSLOT=*functionslot*
When REQUEST=SETFUNCENBL is specified, a required input parameter that identifies the slot in the product area that is holding the data for the function entry that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,FUNCTIONSLOT=*functionslot*
When REQUEST=UPDFUNCUSE is specified, a required input parameter that identifies the slot in the product area that is holding the data for the function entry that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,FUNCTIONUPDTYPE=ANYAUTH
,FUNCTIONUPDTYPE=AUTHONLY
,FUNCTIONUPDTYPE=VALUE
When REQUEST=SETFUNCENBL is specified, a required parameter, which indicates in which set of function entries in the selected product area to look for the specified function.

,FUNCTIONUPDTYPE=ANYAUTH
selects the set which authorized and unauthorized callers are allowed to update.

,FUNCTIONUPDTYPE=AUTHONLY
selects the set which only authorized callers are allowed to update.

,FUNCTIONUPDTYPE=VALUE
indicates that the selection of function set is specified as runtime value via parameter FUNCUPDTYPVALUE.

,FUNCTIONUPDTYPE=ANYAUTH
,FUNCTIONUPDTYPE=AUTHONLY
,FUNCTIONUPDTYPE=VALUE
When REQUEST=UPDFUNCUSE is specified, a required parameter, which indicates in which set of function entries in the selected product area to look for the specified function.

,FUNCTIONUPDTYPE=ANYAUTH
selects the set which authorized and unauthorized callers are allowed to update.

,FUNCTIONUPDTYPE=AUTHONLY
selects the set which only authorized callers are allowed to update.

,FUNCTIONUPDTYPE=VALUE
indicates that the selection of function set is specified as runtime value via parameter FUNCUPDTYPVALUE.

,FUNCUPDTYPVALUE=*funcupdtypvalue*
When FUNCTIONUPDTYPE=VALUE and REQUEST=SETFUNCENBL are specified, a required input parameter, which indicates in which set of function entries in the selected product area to look for the specified function:

- specify 0 (zero, see also equate *_XFUNCTIONUPDTYPE_ANYAUTH) to select the set which authorized and unauthorized callers are allowed to update.
- specify 1 (one, see also equate *_XFUNCTIONUPDTYPE_AUTHONLY) to select the set which only authorized callers are allowed to update.

To code: Specify the RS-type address of a one-byte field.

,FUNCUPDTYPVALUE=*funcupdtypvalue*
When FUNCTIONUPDTYPE=VALUE and REQUEST=UPDFUNCUSE are specified, a required input parameter, which indicates in which set of function entries in the selected product area to look for the specified function:

- specify 0 (zero, see also equate *_XFUNCTIONUPDTYPE_ANYAUTH) to select the set which authorized and unauthorized callers are allowed to update.
- specify 1 (one, see also equate *_XFUNCTIONUPDTYPE_AUTHONLY) to select the set which only authorized callers are allowed to update.

To code: Specify the RS-type address of a one-byte field.

,INSTANCEID=*instanceid*
,INSTANCEID=ANY_INSTANCEID
When REQUEST=SETFUNCENBL is specified, an optional input parameter that optionally qualifies the desired product further, for example via a subsystem name, especially in the case of multiple instances of the product on the system. If INSTANCEID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_INSTANCEID.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,INSTANCEID=*instanceid*
,INSTANCEID=ANY_INSTANCEID
When REQUEST=UPDFUNCUSE is specified, an optional input parameter that optionally qualifies the desired product further, for example via a subsystem name, especially in the case of multiple instances of the product on the system. If INSTANCEID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_INSTANCEID.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of FXECNTRL in the following order:

- Use FXECNTRL ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use FXECNTRL ...MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
- Use FXECNTRL ...MF=(E,list-addr,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,PRODAREAADDR=*prodareaaddr*
When REQUEST=APPLYIPLPARM is specified, a required input parameter of the product area to apply any matching deferred-apply statements to. IBM recommends to point the FXECNTRL REQUEST=APPLYIPLPARM to a product area just before the area gets linked into the Function Registry infrastructure, not after.

Note:

- Only this one, directly referenced product area is used for this request. Any further product area instances, as optionally linked via the product area's FXEFRPA_NextInstanceAddr field, are not processed automatically and require their own, separate FXECNTRL REQUEST=APPLYIPLPARM calls.
- The product area is expected to reside in common storage.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,PRODUCTAREA=*productarea*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that is the product area to apply any matching deferred-apply statements to. IBM recommends to point the FXECNTRL REQUEST=APPLYIPLPARM to a product area just before the area gets linked into the Function Registry infrastructure, not after.

Note:

- Only this one, directly referenced product area is used for this request. Any further product area instances, as optionally linked via the product area's FXEFRPA_NextInstanceAddr field, are not processed automatically and require their own, separate FXECNTRL REQUEST=APPLYIPLPARM calls.
- The product area is expected to reside in common storage.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight character field.

,PRODUCTID=*productid*
,PRODUCTID=ANY_PRODUCTID
When REQUEST=SETFUNCENBL is specified, an optional input parameter that qualifies the desired product further, for example via an FMID, especially in the case of multiple instances of the product on the system. If PRODUCTID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_PRODUCTID.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,PRODUCTID=*productid*
,PRODUCTID=ANY_PRODUCTID
When REQUEST=UPDFUNCUSE is specified, an optional input parameter that qualifies the desired product further, for example via an FMID, especially in the case of multiple instances of the product on the system. If PRODUCTID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_PRODUCTID.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,PRODUCTNAME=*productname*
When REQUEST=SETFUNCENBL is specified, a required input parameter which contains the name of the product that is to be used for the requested operation. If PRODUCTNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,PRODUCTNAME=*productname*
When REQUEST=UPDFUNCUSE is specified, a required input parameter which contains the name of the product that is to be used for the requested operation. If PRODUCTNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,PRODUCTSLOT=*productslot*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that identifies the product slot, in the specified vendor area, which will be, or already is, holding the pointer to the specified product area. Slot numbers are 1-based, meaning the first slot is slot number 1.

You must specify both PRODUCTSLOT and the PRODAREAADDR or PRODUCTAREA parameter (which implies the product name, instance ID and product ID information) to ensure proper matching against either product slot or product name syntax, as chosen by the user in the deferred-apply statements from the FXEPRMxx PARMLIB members identified by the system parameter FXE.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,PRODUCTSLOT=*productslot*
When REQUEST=SETFUNCENBL is specified, a required input parameter that identifies the slot in the selected vendor area that is holding the pointer to the product area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,PRODUCTSLOT=*productslot*
When REQUEST=UPDFUNCUSE is specified, a required input parameter that identifies the slot in the selected vendor area that is holding the pointer to the product area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

REQUEST=APPLYIPLPARM
REQUEST=SETFUNCENBL
REQUEST=UPDFUNCUSE
A required parameter, which identifies the type of request.

REQUEST=SETFUNCENBL
Set the enablement state for a function of a given product, owned by a given vendor. Wildcard use in the selectors is allowed and FXECNTRL will attempt to update all matching function entries.

REQUEST=UPDFUNCUSE
Update the usage information for a function of a given product, owned by a given vendor, based on the current function usage indicator field FXEFRFE_Used of the selected function entry:

- If FXEFRFE_Used=FXEFRFE_Used_NO_Num, then field FXEFRFE_Used is set to FXEFRFE_Used_YES_Num, indicating that the function has been used at least once.
- If FXEFRFE_Used=FXEFRFE_Used_YES_Num, field FXEFRFE_Used will not be changed and it will continue to indicate that the function has been used at least once.
- If FXEFRFE_Used=FXEFRFE_Used_USE_OTHER_Num, then field FXEFRFE_UsageCount will be incremented by one.
- If FXEFRFE_Used=FXEFRFE_Used_NOT_TRACKED_Num, then the request ends with a warning since the function owner set this function up to not have any usage tracked. This function entry will not be updated.

Wildcard use in the selectors is allowed and FXECNTRL will attempt to update all matching function entries.

REQUEST=APPLYIPLPARM
Apply any matching deferred-apply statements to the function entries of the single product area specified by the PRODUDTAREA or PRODAREAADDR parameter. These statements are from FXEPRMxx PARMLIB members that were identified by the system parameter FXE at system start time, but which could not be applied at that time yet, since the Function Registry infrastructure had not been fully established. IBM recommends to call FXECNTRL REQUEST=APPLYIPLPARM for newly created product areas just before such a product area gets added to the Function Registry infrastructure to ensure that the product area is properly primed, before any end user, or other product code, can start modifying contained function entries via interfaces like the SETFXE command, the SET FXE command, the FXECNTRL service or via direct memory access.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,VENDORNAME=*vendorname*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that contains the vendor name from the vendor area which will be, or already is, referencing the specified product area.

Both VENDORNAME and VENDORSLOT parameters have to be specified to ensure proper matching against either vendor name or vendor slot syntax, as chosen by the user in the deferred-apply statements from the FXEPRMxx PARMLIB members identified by the system parameter FXE.

The VENDORNAME value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.

- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,VENDORNAME=*vendorname*
When REQUEST=SETFUNCENBL is specified, a required input parameter which contains the name of the vendor that is to be used for the requested operation. If VENDORNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,VENDORNAME=*vendorname*
When REQUEST=UPDFUNCUSE is specified, a required input parameter which contains the name of the vendor that is to be used for the requested operation. If VENDORNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,VENDORSLOT=*vendorslot*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that identifies the slot in the anchor table that will be, or already is, holding the pointer to the vendor area which will be referencing the specified product area. Slot numbers are 1-based, meaning the first slot is slot number 1.

Both VENDORSLOT and VENDORNAME parameters have to be specified to ensure proper matching against either vendor slot or vendor name syntax, as chosen by the user in the deferred-apply statements from the FXEPRMxx PARMLIB members identified by the system parameter FXE.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,VENDORSLOT=*vendorslot*
When REQUEST=SETFUNCENBL is specified, a required input parameter that identifies the slot in the anchor table that is holding the pointer to the vendor area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,VENDORSLOT=*vendorslot*
When REQUEST=UPDFUNCUSE is specified, a required input parameter that identifies the slot in the anchor table that is holding the pointer to the vendor area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

ABEND Codes

None.

Return and Reason codes

When the FXECNTRL macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro FXEZCTRL provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the *xxxx* value.

Table 20. Return and reason codes for the FXECNTRL macro

Return code	Reason code	Equate symbol, meaning and action
0	–	**Equate symbol**: FxeCntrlRc_OK **Meaning**: Successfully executed the request. **Action**: None required
4	–	**Equate symbol**: FxeCntrlRc_Warning **Meaning**: Warning **Action**: Refer to action under the individual reason code.
4	*xxxx*04F1	**Equate symbol**: FxeCntrlRsn_VendorNotFound **Meaning**: Requested function entry not found: No vendor matched. **Action**: Ensure that the given vendor, product, and function selectors identify an existent entry in the IBM Function Registry for z/OS.
4	*xxxx*04F2	**Equate symbol**: FxeCntrlRsn_ProductNotFound **Meaning**: Requested function entry not found: None of the matching vendors had a matching product. **Action**: Ensure that the given vendor, product, and function selectors identify an existent entry in the IBM Function Registry for z/OS.
4	*xxxx*04F3	**Equate symbol**: FxeCntrlRsn_FunctionNotFound **Meaning**: Requested function entry not found: None of the matching products had a matching function entry. **Action**: Ensure that the given vendor, product, and function selectors identify an existent entry in the IBM Function Registry for z/OS.
8	–	**Equate symbol**: FxeCntrlRc_Error **Meaning**: Error **Action**: Refer to action under the individual reason code.
8	*xxxx*0801	**Equate symbol**: FxeCntrlRsn_BadEnvNotEnabled **Meaning**: Not enabled. **Action**: Avoid using FXECNTRL when not enabled for I/O and external interrupts.
8	*xxxx*0802	**Equate symbol**: FxeCntrlRsn_BadEnvLocked **Meaning**: Locked. **Action**: Avoid using FXECNTRL when a lock is held.
8	*xxxx*0803	**Equate symbol**: FxeCntrlRsn_BadEnvSrbmode **Meaning**: SRB mode. **Action**: Avoid issuing FXECNTRL in SRB mode.

Table 20. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	*xxxx*0804	**Equate symbol**: FxeCntrlRsn_BadEnvFRR **Meaning**: The caller had an EUT FRR established. **Action**: Avoid using FXECNTRL when an EUT FRR is established.
8	*xxxx*0805	**Equate symbol**: FxeCntrlRsn_BadParmlist **Meaning**: Error accessing parameter list. **Action**: Make sure that the provided parameter list is valid.
8	*xxxx*0806	**Equate symbol**: FxeCntrlRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure that the ALET associated with the parameter list is valid. The access register might not have been set up correctly.
8	*xxxx*0807	**Equate symbol**: FxeCntrlRsn_BadVendorName **Meaning**: Error accessing VENDORNAME. **Action**: Make sure that the provided VENDORNAME parameter is valid.
8	*xxxx*0808	**Equate symbol**: FxeCntrlRsn_BadVendorNameALET **Meaning**: Bad VENDORNAME ALET. **Action**: Make sure that the ALET associated with the VENDORNAME parameter is valid. The access register might not have been set up correctly.
8	*xxxx*0809	**Equate symbol**: FxeCntrlRsn_BadProductName **Meaning**: Error accessing PRODUCTNAME. **Action**: Make sure that the provided PRODUCTNAME parameter is valid.
8	*xxxx*080A	**Equate symbol**: FxeCntrlRsn_BadProductNameALET **Meaning**: Bad PRODUCTNAME ALET. **Action**: Make sure that the ALET associated with the PRODUCTNAME parameter is valid. The access register might not have been set up correctly.
8	*xxxx*080B	**Equate symbol**: FxeCntrlRsn_BadProductID **Meaning**: Error accessing PRODUCTID. **Action**: Make sure that the provided PRODUCTID parameter is valid.
8	*xxxx*080C	**Equate symbol**: FxeCntrlRsn_BadProductIdALET **Meaning**: Bad PRODUCTID ALET. **Action**: Make sure that the ALET associated with the PRODUCTID parameter is valid. The access register might not have been set up correctly.
8	*xxxx*080D	**Equate symbol**: FxeCntrlRsn_BadInstanceID **Meaning**: Error accessing INSTANCEID. **Action**: Make sure that the provided INSTANCEID parameter is valid.

Table 20. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	*xxxx*080E	**Equate symbol**: FxeCntrlRsn_BadInstanceIdALET **Meaning**: Bad INSTANCEID ALET. **Action**: Make sure that the ALET associated with the INSTANCEID parameter is valid. The access register might not have been set up correctly.
8	*xxxx*080F	**Equate symbol**: FxeCntrlRsn_BadFunctionName **Meaning**: Error accessing FUNCTIONAME. **Action**: Make sure that the provided FUNCTIONAME parameter is valid.
8	*xxxx*0810	**Equate symbol**: FxeCntrlRsn_BadFunctionNameALET **Meaning**: Bad FUNCTIONNAME ALET. **Action**: Make sure that the ALET associated with the FUNCTIONNAME parameter is valid. The access register might not have been set up correctly.
8	*xxxx*0811	**Equate symbol**: FxeCntrlRsn_BadRequest **Meaning**: A bad REQUEST type has been specified. **Action**: Use one of the supported request types.
8	*xxxx*0812	**Equate symbol**: FxeCntrlRsn_BadParmlistVersion **Meaning**: The specified version of the macro is not compatible with the current version of IBM Function Registry for z/OS. **Action**: Avoid requesting parameters that are not supported by this version of IBM Function Registry for z/OS.
8	*xxxx*0813	**Equate symbol**: FxeCntrlRsn_BadFunctionUpdateType **Meaning**: Bad function update type. **Action**: Use only supported updated types: ANYAUTH, AUTHONLY, or VALUE. If VALUE, then FUNCUPDTYPVALUE has to be 0 (ANYAUTH) or 1 (AUTHONLY).
8	*xxxx*0814	**Equate symbol**: FxeCntrlRsn_NotAuthorizedForFuncUpdType **Meaning**: Not authorized for FUNCTIONUPDTYPE. **Action**: Only authorized callers are allowed to use FUNCTIONUPDTYPE=AUTHONLY or FUNCTIONUPDTYPE=VALUE and FUNCUPDTYPVALUE=1 (AUTHONLY).
8	*xxxx*0815	**Equate symbol**: FxeCntrlRsn_BadEnabledValue **Meaning**: Bad ENABLED value. **Action**: Use only supported ENABLED values: YES, NO, or VALUE. If VALUE, then ENABLEDVALUE has to be 1 (YES) or 0 (NO).

Table 20. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	*xxxx*0816	**Equate symbol**: FxeCntrlRsn_EnabledNotTracked **Meaning**: Enablement state is not tracked. The request was aborted and no matching function entries had their enablement state changed. If wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all matching function entries had their enablement state changed. **Action**: The selected function entry is set up to not track enablement state. Refer to the documentation for this function entry for more information about its capabilities.
8	*xxxx*0817	**Equate symbol**: FxeCntrlRsn_UsageNotTracked **Meaning**: Usage information is not tracked. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their usage information changed. **Action**: The selected function entry is set up to not track usage information. Refer to the documentation for this function entry for more information about its capabilities.
8	*xxxx*0818	**Equate symbol**: FxeCntrlRsn_BadVendorSlot **Meaning**: Invalid VENDORSLOT. **Action**: Ensure a valid VENDORSLOT number is specified. In particular 0 (zero) is not valid. Slot numbers start at 1 (one).
8	*xxxx*0819	**Equate symbol**: FxeCntrlRsn_BadProductSlot **Meaning**: Invalid PRODUCTSLOT. **Action**: Ensure a valid PRODUCTSLOT number is specified. In particular 0 (zero) is not valid. Slot numbers start at 1 (one).
8	*xxxx*081A	**Equate symbol**: FxeCntrlRsn_BadFunctionSlot **Meaning**: Invalid FUNCTIONSLOT. **Action**: Ensure a valid FUNCTIONSLOT number is specified. In particular 0 (zero) is not valid. Slot numbers start at 1 (one).
8	*xxxx*081B	**Equate symbol**: FxeCntrlRsn_NotAuthorizedForApplyIplParm **Meaning**: Not authorized for REQUEST=APPLYIPLPARM. **Action**: Only authorized callers are allowed to use request type APPLYIPLPARM.
8	*xxxx*081C	**Equate symbol**: FxeCntrlRsn_BadProductArea **Meaning**: Error accessing product area. **Action**:Make sure that the provided product area, referenced by parameter PRODUCTAREA or PRODAREAADDR, is valid (in common storage, writable, not NULL for example).
8	*xxxx*081D	**Equate symbol**: FxeCntrlRsn_BadParmlistUpd **Meaning**: Error updating parameter list for output parameters. **Action**: Make sure that the provided parameter list is valid and writable.

Table 20. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
0C	–	**Equate symbol**: FxeCntrlRc_SevereError **Meaning**: Severe Error / Environment Error **Action**: Refer to action under the individual reason code.
0C	*xxxx*0C80	**Equate symbol**: FxeCntrlRsn_RegistryNotAvailable **Meaning**: IBM Function Registry for z/OS is not available. **Action**: This might be a temporary situation in the very early phases of the system start. Contact IBM support if this problem persists and if problem reporting databases do not list an existing fix.
0C	*xxxx*0C81	**Equate symbol**: FxeCntrlRsn_EnabledValNotSupported **Meaning**: Unsupported ENABLED value in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their enablement state changed. **Action**: Contact the owner of the selected function entry. An unsupported enabled value was found in the registry for this function entry.
0C	*xxxx*0C82	**Equate symbol**: FxeCntrlRsn_EnabledValLocked **Meaning**: ENABLED locked in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their enablement state changed. **Action**: The selected function entry is not set up to have its enablement state changed. Refer to the documentation for this function entry for more information about its capabilities.
0C	*xxxx*0C83	**Equate symbol**: FxeCntrlRsn_UsageValNotSupported **Meaning**: Unsupported usage indicator in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their usage indicator changed. **Action**: Contact the owner of the selected function entry. An unsupported usage indicator was found in the registry for this function entry.
0C	*xxxx*0C8A	**Equate symbol**: FxeCntrlRsn_EnabledInvalidFuncEntry **Meaning**: Inconsistent function entry (FXEFRFE) in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their enablement state changed. **Action**: Contact the owner of the selected function entry. An inconsistent function entry was found in the registry. For example, the function entry did not have a recognized acronym value set in its FXEFRFE_ID field.

Table 20. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
0C	*xxxx*0C8B	**Equate symbol**: FxeCntrlRsn_UsageInvalidFuncEntry **Meaning**: Inconsistent function entry (FXEFRFE) in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their function usage indicator changed. **Action**: Contact the owner of the selected function entry. An inconsistent function entry was found in the registry. For example, the function entry did not have a recognized acronym value set in its FXEFRFE_ID field.
10	–	**Equate symbol**: FxeCntrlRc_CompError **Meaning**: Component error. **Action**: Report the associated reason code to the system programmer to contact IBM Service.

Chapter 18. GETDSAB — Accessing the DSAB chain

Description

The GETDSAB macro returns a pointer to the data set association block (DSAB) associated with a DD name.

Use the GETDSAB macro to:

- Retrieve the address of the first DSAB associated with a DD name, as specified by:
  - An input DD name
  - An input device control block (DCB) address
  - An input task control block (TCB) address.
- Scan the DSAB chain

See *z/OS MVS Programming: Authorized Assembler Services Guide* for procedures that use the DSAB address returned by the GETDSAB macro to obtain the address of the TIOT entry and the UCB address.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state or supervisor state, and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit addressing mode
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

To use GETDSAB, the caller must include the DSAB mapping macro (IHADSAB), the CVT mapping macro (CVT), and the JESCT mapping macro (IEFJESCT).

The caller must provide or inherit serialization on the SYSZTIOT resource before calling the GETDSAB macro and while using the output addresses of the macro. The minimum required level of serialization is shared (SHR).

The GETDSAB service does not provide a recovery environment. Because the service runs in task mode, the system uses any recovery environment that is defined to the caller before invoking GETDSAB.

Restrictions

Use caution when running as a system user.

GETDSAB uses the current JSCB and not the active JSCB.

GETDSAB facilitates the process of finding a DSAB for an end user. In this case, the end user runs with TCBJSCB pointing to the active JSCB.

System users who are in the window before or after the jobstep program is attached run the risk of using the current JSCB, which might differ from the active JSCB. System address spaces might have the same current and active JSCB address.

Register information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

On input to the macro, register 13 must contain the address of an 18-word save area.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code if GPR 15 contains 12; otherwise, used as a work register by the system

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

Performance implications

There are no performance implications related to GETDSAB.

Syntax

The standard form of the GETDSAB macro follows.

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede GETDSAB.
GETDSAB	
␣	One or more blanks must follow GETDSAB.
FIRST	
NEXT	
DCBPTR=*dcb addr*	*dcb addr*: RX-type address, or register (2) - (12).

Syntax	Description
DDNAME=*dd addr*	*dd addr*: RX-type address, or register (2) - (12). This address specifies an 8-byte field which contains a DD name.
,DSABPTR=*dsab addr*	*dsab addr*: RX-type address, or register (2) - (12).
,TCBPTR=*tcb addr*	*tcb addr*: RX-type address, or register (2) - (12). **Default**: TCBPTR=0
,RETCODE=*retcode addr*	*retcode addr*: RX-type address, or register (2) - (12) of fullword output variable
,RSNCODE=*rsn addr*	*name*: RX-type address, or register (2) - (12).
,LOC=BELOW	**Default** = BELOW
,LOC=ANY	

Parameters

The parameters are described as follows:

FIRST
NEXT
DCBPTR=*dcb addr*
DDNAME=*dd addr*

FIRST requests the first DSAB in the DSAB chain. The system uses the DSAB chain associated with the TCB specified by the TCBPTR parameter, or, if none is specified, by the current TCB.

NEXT requests the pointer to the next DSAB in the DSAB chain, following the one pointed to by the initial value in DSABPTR.

DCBPTR=*dcb addr* specifies the name of a pointer that contains the address of a fullword field. The fullword points to the DCB associated with a DD name. The system retrieves the DSAB pointer associated with the DCB.

When DCBPTR points to an open DCB, DCBPTR and TCBPTR are mutually exclusive.

Do not use the DCBPTR option for a DCB in a DCB OPEN exit, DCB ABEND exit, data management ABEND installation exit or the DCB OPEN installation exit.

When DCBPTR points to a closed DCB, the system selects the DSAB chain associated with the TCB specified by TCBPTR parameter, or, if none is specified, by the current TCB.

DDNAME=*dd addr* specifies a DD name associated with a DSAB. The system puts the address of the DSAB associated with this DD name into the fullword field specified by the DSABPTR parameter. The DSAB selected is that associated with the TCB specified by the TCBPTR parameter, or, if none is specified, by the current TCB. The *dd addr* is an 8-character, left-justified field, with trailing blanks. The *dd addr* may not contain all blanks.

,DSABPTR=*dsab addr*

Specifies the name of a required fullword field that will be set to the address of the desired DSAB.

When used with the NEXT keyword, DSABPTR must contain the address of a DSAB that was previously obtained by invoking GETDSAB with FIRST, DCBPTR, or DDNAME. The system will replace this initial address with the address of the next DSAB in the DSAB chain.

When used with the keywords FIRST, DCBPTR, or DDNAME, DSABPTR is an output field only.

Upon output, DSABPTR contains the address of the specified DSAB if the return code is zero. If the return code is not zero, DSABPTR contains 0.

,TCBPTR=*tcb addr*
Specifies the name of a pointer that contains the address of the TCB associated with the task for which DSAB information is requested.

When DCBPTR points to an open DCB, DCBPTR and TCBPTR are mutually exclusive.

The default, TCBPTR=0, requests the current TCB.

,RETCODE=*retcode addr*
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsncode addr*
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

LOC=BELOW
LOC=ANY
Specifies whether or not GETDSAB should limit where it searches to find the DSAB corresponding to the input DDname.

LOC=BELOW, which is the default, searches only those DSABs residing below the 16 MB line

LOC=ANY searches both below and above the 16 MB line.

Return and reason codes

When control returns from GETDSAB, GPR 15 (and *retcode addr*, if you coded RETCODE) contains one of the following decimal return codes.

Table 21. Return Codes for the GETDSAB Macro

Decimal Return Code	Meaning
00	**Meaning**: Successful completion
04	**Meaning**: Request failed. NEXT was specified when DSABPTR pointed to the last DSAB in the DSAB chain.
08	**Meaning**: Request failed. The specified DSAB was not found.
12	**Meaning**: Request failed. Input values were in error or in conflict.
16	**Meaning**: Request failed. The GETDSAB function is not currently installed on the system. Consult your system programmer.

When control returns from GETDSAB, GPR 0 (and *rsncode addr*, if you coded RSNCODE) might contain one of the following decimal reason codes:

Table 22. Return and Reason Codes for the GETDSAB Macro

Decimal Return Code	Decimal Reason Code	Meaning
12	1200	Request failed because of input error. The DDNAME specified or obtained was all blanks.
12	1210	Request failed because of input error. TCBPTR was specified when DCBPTR points to an open DCB.
12	1220	Request failed because of input error. The DSAB pointed to by DSABPTR is not valid.
12	1230	Request failed because of input error. The LOC=BELOW is requested, but the DSAB chain contains DSABs from both above and below the 16 MB line. Use LOC=ANY for this DSAB chain.

Example 1

In this example, MYDSAB will contain the address of DSAB associated with the DD named DD09.

```
            MVC   THEDD,=CL8'DD09'
            GETDSAB DDNAME=THEDD,DSABPTR=MYDSAB
             .
             .
             .
AUTO        DSECT
THEDD       DS    CL8
MYDSAB      DS    AL4
```

Example 2

In this example, the first invocation of GETDSAB will set MYDSAB to the address of the first DSAB in the DSAB chain. MYRC will contain the return code.

The second invocation of GETDSAB will replace the initial address in MYDSAB with the address of the next DSAB in the DSAB chain.

```
            GETDSAB FIRST,DSABPTR=MYDSAB,RETCODE=MYRC
             .
             .
             .
            GETDSAB NEXT,DSABPTR=MYDSAB
             .
             .
             .
AUTO        DSECT
MYDSAB      DS    AL4
MYRC        DS    F
```

This technique can be used to get the DSAB for the first DD in a concatenation and then to step through the DSABs for all other DDs in the concatenation. It is the user's responsibility to determine when the DSAB for the last DD in the concatenation has been fetched, because a subsequent invocation of GETDSAB NEXT will simply return the next DSAB on the chain (if one exists), even if it is for a different DD statement.

Example 3

In this example, DCBPTR contains the address of a fullword pointer that points to the DCB associated with a DD name. MYDSAB will contain the address of the DSAB associated with the DCB. MYRSN will contain the reason code.

```
            GETDSAB DCBPTR=MYDCB,DSABPTR=MYDSAB,RSNCODE=MYRSN
             .
             .
             .
AUTO        DSECT
MYDSAB      DS    AL4
MYDCB       DS    AL4
MYRSN       DS    F
```

Example 4

If DCBPTR points to an open DCB, DCBPTR and TCBPTR are mutually exclusive. The request will fail with return code 12. MYDSAB will contain 0.

If DCBPTR points to a closed DCB, the system will search the DSAB chain associated with the TCB. MYDSAB will contain the address of the DSAB related to the TCB specified by TCBPTR.

```
            GETDSAB DCBPTR=MYDCB,DSABPTR=MYDSAB,TCBPTR=MYTCB
             .
             .
             .
AUTO        DSECT
MYDSAB      DS    AL4
```

```
MYDCB     DS   AL4
MYTCB     DS   F
```

GETDSAB - List form

Use the list form of the GETDSAB macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

The list form of the GETDSAB macro follows.

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GETDSAB.
GETDSAB	
ƀ	One or more blanks must follow GETDSAB.
MF=(L,*stor addr*)	*stor addr*: symbol.
MF=(L,*stor addr,attr*)	*attr*: 1- to 60-character input string. **Default**: 0D

Parameters

The following parameters are the only ones you can specify using the list format:

MF=L
Specifies the list form of the GETDSAB macro.

The *stor addr* parameter specifies the name of a required storage area for the parameter list. This storage area will be generated as part of the macro expansion and should not be separately defined by the user. Note also, that the "*stor addr*" in the List and Execute forms of the macro must refer to the same storage area.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

GETDSAB - Execute form

Use the execute form of the GETDSAB macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the GETDSAB macro follows.

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GETDSAB.
GETDSAB	
ƀ	One or more blanks must follow GETDSAB.
FIRST	
NEXT	
DCBPTR=*dcb addr*	*dcb addr*: RX-type address, or register (2) - (12).
DDNAME=*dd addr*	*dd addr*: RX-type address, or register (2) - (12).
,DSABPTR=*dsab addr*	*dsab addr*: RX-type address, or register (2) - (12).
,TCBPTR=*tcb addr*	*tcb addr*: RX-type address, or register (2) - (12). **Default**: TCBPTR=0
,RETCODE=*retcode addr*	*retcode addr*: RX-type address, or register (2) - (12) of fullword output variable
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address, or register (2) - (12).
,LOC=BELOW	**Default** = BELOW
,LOC=ANY	
MF=(E,*stor addr*)	*stor addr*: RX-type address, or any register (1) - (12). If register 1 is specified, its value may be changed by the macro invocation.
MF=(E,*stor addr*,COMPLETE)	**Default**: COMPLETE

Parameters

The parameters are explained under the standard form of the GETDSAB macro, with the following exception:

MF=(E,*stor addr*)
MF=(E,*stor addr*,COMPLETE)
Specifies the execute form of the macro.

The *stor addr* parameter specifies the name of a required storage area for the parameter list.

The COMPLETE parameter specifies the degree of macro parameter syntax checking. COMPLETE checks for required macro keywords and supplies defaults for optional parameters that are not specified.

Chapter 19. GETMAIN — Allocate virtual storage

Description

Use the GETMAIN macro to request one or more areas of virtual storage.

Before obtaining storage, be sure to read the topic about selecting the right subpool for virtual storage requests in *z/OS MVS Programming: Authorized Assembler Services Guide*.

You can also use the STORAGE macro to obtain storage. Compared to GETMAIN, STORAGE provides an easier-to-use interface and has fewer restrictions and locking requirements. See the virtual storage management chapter in *z/OS MVS Programming: Authorized Assembler Services Guide* for a comparison of GETMAIN and STORAGE.

The GETMAIN macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the BRANCH and OWNER parameters.

Note:

1. When you obtain storage, the system clears the requested storage to zeros if you obtain either:
   - 8192 bytes or more from a pageable, private storage subpool.
   - 4096 bytes or more from a pageable, private storage subpool, with BNDRY=PAGE or STARTBDY=12 specified.

   In all other cases you must not assume that the storage is cleared to zeros.

   The caller can specify CHECKZERO=YES to detect these and other cases where the system clears the requested storage to zeros.
2. Do not allocate user key (8-15) storage in the common area because it can be read or written by any program in any address space.

The GETMAIN macro provides two types of entry linkage: SVC entry and branch entry. If you do not specify the BRANCH parameter, the GETMAIN service receives control through **SVC entry**. If you specify the BRANCH parameter, the GETMAIN service receives control through **branch entry**.

If you use GETMAIN to request real storage backing above 2 gigabytes, but your system does not support 64-bit storage, your request will be treated as a request for backing above 16 megabytes, even on earlier releases of z/OS that do not support backing above 2 gigabytes. However, boundary requirements indicated by the CONTBDY and STARTBDY parameters will be ignored by earlier releases of z/OS.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For subpools 0-127: problem state and PSW key 8-15. For subpools 131 and 132, **one or more of the following**: • Supervisor state • PSW key 0-7 • APF-authorization. • PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be obtained. For other subpools: **one or more of the following**: • Supervisor state • PSW key 0-7 • APF-authorization. For branch entry: supervisor state and PSW key 0.
Dispatchable unit mode:	For SVC entry: task. For branch entry: task or SRB.
Cross memory mode:	For SVC entry: PASN=HASN=SASN. For branch entry: any PASN, any HASN, any SASN.
AMODE:	For SVC entry: 24- or 31- or 64-bit. For branch entry: 24- or 31-bit. • For R, LC, LU, VC, VU, EC, or EU requests: If the calling program is in 31-bit mode, the system treats all addresses and values as 31-bit. Otherwise, the system treats addresses and values as 24-bit. • For RC, RU, VRC, and VRU requests: The system treats all addresses and values as 31-bit.
RMODE:	For SVC-entry, includes 64-bit
ASC mode:	For BRANCH=(YES,GLOBAL): primary or access register (AR). For all other requests: primary. Callers in AR mode must use BRANCH=(YES,GLOBAL) and can obtain only global (common) storage.
Interrupt status:	For BRANCH=(YES,GLOBAL): disabled for I/O and external interrupts. For all other requests: enabled for I/O and external interrupts.
Locks:	• For SVC entry: no locks may be held. • For BRANCH=YES: your program must hold the local lock for the currently addressable address space. This must be the address space from which the storage is to be obtained. • For BRANCH=YES, when running in cross-memory mode: your program must hold the CML lock for the currently addressable address space. This must be the address space from which the storage is to be obtained. • For BRANCH=(YES,GLOBAL): your program must be in an z/OS-recognized state of disablement, which can be attained by obtaining the CPU lock.

Environmental factor	Requirement
Control parameters:	For LC, LU, VC, VU, EC, EU requests: control parameters must be in the primary address space. For other requests: control parameters are in registers.

Programming requirements

Before issuing the GETMAIN macro in AR mode, issue SYSSTATE ASCENV=AR.

Restrictions

- For SVC entry, the caller cannot have an EUT FRR established.

Input register information for SVC entry

Before issuing the GETMAIN macro without the BRANCH parameter (SVC entry) the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information for SVC entry

For LC, LU, VC, VU, EC, and EU requests: when control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

For RC, RU, and R requests: when control returns to the caller the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

For VRC and VRU requests: when control returns to the caller the GPRs contain:

Register
Contents

0
For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Input register information for BRANCH=YES

Before issuing the GETMAIN macro with BRANCH=YES, the caller must ensure that the following GPRs contain the specified information:

Register
Contents

4
The address of the input TCB, if you are obtaining private storage.

Set GPR 4 to 0 or the address of a TCB in the currently addressable address space. Setting the GPR 4 to 0 identifies the input TCB as the TCB that owns the cross-memory resources for the currently addressable address space (task whose TCB address is in ASCBXTCB).

For an explanation of the term **input TCB**, and to determine system-assigned defaults for private storage ownership, see the topic about selecting the right subpool for virtual storage requests in *z/OS MVS Programming: Authorized Assembler Services Guide*.

7
The address of the ASCB for the currently addressable address space.

Output register information for BRANCH=YES

For RC, RU, and R requests: when control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2
Unchanged

3
For R requests, unchanged. For RC and RU requests, used as a work register by the system.

4-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

For VRC and VRU requests: when control returns to the caller, the GPRs contain:

Register
Contents

0
For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system. storage obtained.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2
Unchanged

3
Used as a work register by the system.

4-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

For EC, EU, LC, LU, VC, and VU requests: when control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

Input register information for BRANCH=(YES,GLOBAL)

For RC, RU, VRC, and VRU requests (the only valid requests with BRANCH=(YES,GLOBAL)): the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information for BRANCH=(YES,GLOBAL)

For RC and RU requests: when control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2
Unchanged

3-4
Used as work registers by the system.

5-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

For VRC and VRU requests: when control returns to the caller, the GPRs contain:

Register
Contents

0
For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2
Unchanged.

3-4
Used as work registers by the system.

5-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the service returns control.

Performance implications

Repeatedly issuing the GETMAIN macro can slow down performance. If your program requires many identically sized storage areas, use the CPOOL macro or callable cell pool services for better performance.

Syntax

The standard form of the GETMAIN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GETMAIN.
GETMAIN	
ƀ	One or more blanks must follow GETMAIN.
LC,LA=*length addr*,A=*addr*	*length addr*: A-type address, or register (2) - (12).
LU,LA=*length addr*,A=*addr*	*length value*: symbol, decimal number, or register (2) - (12).
VC,LA=*length addr*,A=*addr*	If RC or RU is specified, register (0) may also be specified.
VU,LA=*length addr*,A=*addr*	
EC,LV=*length value*,A=*addr*	*addr*: A-type address or register (2) - (12).
EU,LV=*length value*,A=*addr*	**Note:** RC, RU, VRC, or VRU must be used for address greater than 16 megabytes.
RC,LV=*length value*	
RU,LV=*length value*	

Syntax	Description
R,LV=*length value*	
VRC,LV=*(maximum length value, minimum length value)*	*maximum length value*: symbol, decimal number, or register (2) - (12).
VRU,LV=*(maximum length value, minimum length value)*	*minimum length value*: symbol, decimal number, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr*: symbol or decimal number 0-255; or register (2) - (12).
	Default: SP=0 **Note**: Specify the subpool as follows: • Use the SP parameter for LC, LU, VC, VU, EC, EU, RC, RU, VRC, and VRU requests, and for R requests where LV does not indicate register 0. • Use register 0 for R requests with LV=(0); do not code the SP parameter. The low-order three bytes of register 0 must contain the length of the requested storage, and the high-order byte must contain the subpool number.
,BNDRY=DBLWD	**Default:** BNDRY=DBLWD
,BNDRY=PAGE	**Note:** This parameter may not be specified with R above.
,CONTBDY=*containing_bdy*	*containing_bdy*: Decimal number 3-31 or register (2) - (12). **Note**: CONTBDY may be specified only with RC or RU.
,STARTBDY=*starting_bdy*	*starting_bdy*: Decimal number 3-31 or register (2) - (12). **Note**: STARTBDY may be specified only with RC or RU.
,BRANCH=YES	**Note**: BRANCH=(YES,GLOBAL) may be specified only with RC, RU, VRC, or VRU.
,BRANCH=(YES,GLOBAL)	
,KEY=*key number*	*key number*: decimal numbers 0-15, or register (2) - (12).
	Note: KEY may be specified only with RC, RU, VRC, or VRU.
,LOC=24	**Note**: The LOC parameter applies only when used with RC, RU, VRC, or VRU request types. For all other types, LOC=24 is used.
,LOC=(24,31)	
,LOC=(24,64)	
,LOC=31	
,LOC=(31,31)	
,LOC=(31,64)	

Syntax	Description
,LOC=RES	**Default**: LOC=RES (for request types where LOC applies); LOC=24 (for request types where LOC does not apply)
,LOC=(RES,31)	
,LOC=(RES,64)	
,LOC=EXPLICIT	**Note**: You must specify the INADDR parameter with EXPLICIT.
,LOC=(EXPLICIT,24)	
,LOC=(EXPLICIT,31)	
,LOC=(EXPLICIT,64)	
,INADDR=*stor addr*	*stor addr*: RX-type address or register (1)-(12). **Note**: This parameter can only be specified with LOC=EXPLICIT.
,OWNER=HOME	**Default:** OWNER=HOME
,OWNER=PRIMARY	
,OWNER=SECONDARY	
,OWNER=SYSTEM	
,CHECKZERO=YES	**Default:** CHECKZERO=NO
,CHECKZERO=NO	**Note**: CHECKZERO may be specified only with RC, RU, VRC, or VRU.
,RELATED=*value*	*value:* Any valid assembler character string

Parameters

The parameters are explained as follows.

The first parameter of the GETMAIN macro is positional and is required. This parameter describes the type or mode of the GETMAIN request. The first parameter can be one of the following values:

LC,LA=*length addr, A=addr*
LU,LA=*length addr, A=addr*
VC,LA=*length addr, A=addr*
VU,LA=*length addr, A=addr*
EC,LV=*length value, A=addr*
EU,LV=*length value, A=addr*
RC,LV=*length value*
RU,LV=*length value*
R,LV=*length value*
VRC,LV=*(maximum length value,minimum length value)*
VRU,LV=*(maximum length value,minimum length value)*

LC and LU indicate conditional (LC) and unconditional (LU) list requests, and specify requests for one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage

areas is returned in a list beginning at the address specified in the A parameter. No virtual storage is allocated unless all of the requests in the list can be satisfied.

VC and VU indicate conditional (VC) and unconditional (VU) variable requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is between the two values at the address specified in the LA parameter. The address and actual length of the allocated virtual storage area are returned by the system at the address indicated in the A parameter.

EC and EU indicate conditional (EC) and unconditional (EU) element requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is indicated by the parameter, LV=*length value*. The address of the allocated virtual storage area is returned at the address indicated in the A parameter.

RU and R indicate unconditional register requests; RC indicates a conditional register request. RC, RU, and R specify requests for single areas of virtual storage. The length of the single virtual area is indicated by the parameter, LV=*length value*. The address of the allocated virtual storage area is returned in register 1.

VRC and VRU indicate variable register conditional (VRC) and unconditional (VRU) requests for a single area of virtual storage. The length returned will be between the maximum and minimum lengths specified by the parameter LV=(*maximum length value, minimum length value*). The address of the allocated virtual storage is returned in register 1 and the length in register 0.

Note:

1. A **conditional request** indicates that the active unit of work is not to be abnormally terminated if there is insufficient contiguous virtual storage to satisfy the request. A conditional request does not prevent all abnormal terminations. For example, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. An **unconditional request** indicates that the active unit of work is to be abnormally terminated whenever the request cannot complete successfully.
2. The LC, LU, VC, VU, EC, EU, and R requests can be used only to obtain virtual storage with addresses below 16 megabytes. The RC, RU, VRC, and VRU requests can be used to obtain virtual storage with addresses above 16 megabytes.

LA specifies the virtual storage address of consecutive fullwords starting on a fullword boundary. Each fullword must contain the required length in the low-order three bytes, with the high-order byte set to 0. The lengths should be multiples of 8; if they are not, the system uses the next higher multiple of 8. If VC or VU was coded, two words are required. The first word contains the minimum length required, the second word contains the maximum length. If LC or LU was coded, one word is required for each virtual storage area requested; the high-order bit of the last word must be set to 1 to indicate the end of the list. The list must not overlap the virtual storage area specified in the A parameter.

LV=*length value* specifies the length, in bytes, of the requested virtual storage. The number should be a multiple of 8; if it is not, the system uses the next higher multiple of 8. If R is specified, LV=(0) may be coded; the low-order three bytes of register 0 must contain the length, and the high-order byte must contain the subpool number. LV=(*maximum length value, minimum length value*) specifies the maximum and minimum values of the length of the storage request.

The A parameter specifies the virtual storage address of consecutive fullwords, starting on a fullword boundary. The system places the address of the virtual storage area allocated in one or more words. If E was coded, one word is required. If LC or LU was coded, one word is required for each entry in the LA list. If VC or VU was coded, two words are required. The first word contains the address of the virtual storage area, and the second word contains the length actually allocated. The list must not overlap the virtual storage area specified in the LA parameter.

,SP=*subpool nmbr*

Specifies the number of the subpool from which the virtual storage area is to be allocated. If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. Valid subpool numbers range from 0 to 255. See the topic about selecting the right subpool for virtual storage requests in *z/OS MVS Programming: Authorized Assembler Services Guide* for detailed guidance on subpool selection.

Note:

1. Callers running in supervisor state and key zero, who specify subpool 0, will obtain storage from subpool 252. Therefore, when requesting a dump of this storage using the SDUMP or SDUMPX macro, they must specify subpool 252 rather than 0.
2. Requests for storage from subpools 240 and 250 are translated to subpool 0 storage requests.

,BNDRY=DBLWD
,BNDRY=PAGE

Specifies that alignment on a doubleword boundary (DBLWD) or alignment with the start of a virtual page on a 4K boundary (PAGE) is required for the start of a requested area.

If the request specifies one of the LSQA or SQA subpools, the system ignores the BNDRY=PAGE keyword. Requests for storage from these subpools are then fulfilled from a single page, unless the request is greater than a page. See the virtual storage management chapter in *z/OS MVS Programming: Authorized Assembler Services Guide* for a list of LSQA and SQA subpools.

,CONTBDY=*containing_bdy*

Specifies the boundary the obtained storage must be contained within. Specify a power of 2 that represents the containing boundary. Supported values are 3-31. For example, CONTBDY=10 means the containing boundary is 2**10, or 1024 bytes. The containing boundary must be at least as large as the maximum requested boundary. The obtained storage will not cross an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24 - 31 of the register. CONTBDY is valid only with RC or RU.

CONTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

CONTBDY applies to all subpools.

Generally, if you omit this parameter, there is no containing boundary. However, if the GETMAIN is for SQA or LSQA, and is for less than 4 KB, and STARTBDY is specified, the default of CONTBDY is 12, ensuring that the GETMAIN stays within a 4 KB page boundary.

For GETMAIN macros that specify a CONTBDY parameter value that is larger than 12, it is possible that the allocated area spans across a 4 KB page boundary, even when the area is less than or equal to 4 KB and in an SQA or LSQA subpool.

,STARTBDY=*starting_bdy*

Specifies the boundary the obtained storage must start on. Specify a power of 2 that represents the start boundary. Supported values are 3-31. For example, STARTBDY=10 means the start boundary is 2**10, or 1024 bytes. The obtained storage will begin on an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24-31 of the register. STARTBDY is valid only with RC or RU.

STARTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

STARTBDY applies to all subpools.

If you omit this parameter, the start boundary is 8 bytes (equivalent to specifying STARTBDY=3).

,BRANCH=YES
,BRANCH=(YES,GLOBAL)

Specifies that a branch entry is to be used.

BRANCH=YES allows both local (private) and global (common) storage to be allocated. See Input register information for BRANCH=YES for specific information on input register requirements.

BRANCH=(YES,GLOBAL) allows only global storage to be allocated. With BRANCH=(YES,GLOBAL), the SP parameter may designate only subpools 226-228, 231, 239, 241, 245, 247, or 248. BRANCH=(YES,GLOBAL) is valid only with RC, RU, VRC, or VRU.

,KEY=*key number*
Specifies the storage key in which the storage is to be obtained. The valid storage keys are 0-15. If a register is specified, the storage key must be in bits 24-27 of the register. KEY is valid with RC, RU, VRC, or VRU, and applies to subpools 129-132, 227-231, 241, and 249. If you specify KEY without specifying RC, RU, VRC, or VRU, or use KEY for any other subpools, the system ignores the KEY parameter. BRANCH is required with KEY for subpools 227-231, 241, and 249. BRANCH=(YES,GLOBAL) is not valid for subpools 129-132, 229-230, and 249. See the virtual storage management chapter in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about how the system determines the storage key to assign to your storage request.

,LOC=24
,LOC=(24,31)
,LOC=(24,64)
,LOC=31
,LOC=(31,31)
,LOC=(31,64)
,LOC=RES
,LOC=(RES,31)
,LOC=(RES,64)
,LOC=EXPLICIT
,LOC=(EXPLICIT,24)
,LOC=(EXPLICIT,31)
,LOC=(EXPLICIT,64)
Specifies the location of virtual storage and central (also called real) storage. This is especially helpful for callers with 24-bit dependencies. When LOC is specified, central storage is allocated anywhere until the storage is fixed, (for example, using the PGSER macro). You can specify the location of central storage (after the storage is fixed) and virtual storage (whether or not the storage is fixed) using the following LOC parameter values:

LOC=24
Indicates that central and virtual storage are to be located below 16 megabytes.

Note:

1. Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW is still supported, but IBM recommends using LOC=24 instead.
2. LOC=24 should not be used to allocate disabled reference (DREF) storage. If issued in AMODE24, an abend B78 will result. In AMODE31, the LOC=24 parameter will be ignored, and the caller will be given an address above 16 megabytes.
3. For GETMAINs from all SQA subpools, central storage will sometimes be above 16 mg even when LOC=24 or LOC=BELOW is coded.

LOC=(24,31)
Indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=(BELOW,ANY) is the same as specifying LOC=(24,31). LOC=(BELOW,ANY) is still supported, but IBM recommends using LOC=(24,31) instead.

LOC=(24,64)
Indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere in 64-bit storage.

LOC=31
LOC=(31,31)
Indicate that virtual and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC =31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64)
Indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

Note: When you specify LOC=31, the actual location of the virtual storage (that is, whether it is above or below 16 megabytes) depends on the subpool you specify on the SP parameter:

- Some subpools (for example, 203-204) are supported **only above** 16 megabytes. For these subpools, GETMAIN locates virtual storage above 16 megabytes. If you specify LOC=24 for one of these subpools, the system abends your program.

All other subpools are supported both above and below 16 megabytes. For these subpools, specifying LOC=31 causes GETMAIN to try to allocate virtual storage above 16 megabytes. If the attempt fails, GETMAIN tries to allocate virtual storage below 16 megabytes. If this attempt also fails, GETMAIN does not allocate any storage.

When you use LOC=RES to allocate storage that can reside either above or below 16 megabytes, LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be located below 16 megabytes. If the caller resides above 16 megabytes, virtual and central storage are to be located either above or below 16 megabytes.

LOC=(RES,31)
Indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere below 2 gigabytes. In either case, central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=(RES,ANY) is the same as specifying LOC=(RES,31). LOC=(RES,ANY) is still supported, but IBM recommends using LOC=(RES,31) instead.

LOC=(RES,64)
Indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere in 31-bit storage. In either case, central storage can be located anywhere in 64-bit storage.

Note: If your program resides below 16 megabytes but runs with 31-bit addressing mode, you can specify LOC=RES (as a default or explicitly) or LOC=(RES,31) to obtain storage from a subpool supported only above 16 megabytes. Do not specify subpools supported only above 16 megabytes on requests using LOC=RES or LOC=(RES,31) if your program resides below 16 megabytes and runs with 24-bit addressing.

LOC=EXPLICIT
LOC=(EXPLICIT,24)
LOC=(EXPLICIT,31)
LOC=(EXPLICIT,64)
Specify that the requested virtual storage is to be located at the address specified with the INADDR parameter, which is required with EXPLICIT. EXPLICIT is valid only for subpools 0-127, 129-132, 240, 250, 251, and 252. You can use LOC=EXPLICIT only with RC or RU. You cannot specify the BNDRY or OWNER parameters with EXPLICIT.

Note: Specifying LOC=(EXPLICIT,BELOW) is the same as specifying LOC=(EXPLICIT,24). Specifying LOC=(EXPLICIT,ANY is the same as specifying LOC=(EXPLICIT,31). The older specifications are still supported, but IBM recommends using the newer specifications instead.

LOC=(EXPLICIT,31)
Indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere below 2 gigabytes.

LOC=(EXPLICIT,24)
Indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage is to be located below 16 megabytes. The virtual storage address specified on the INADDR parameter must be below 16 megabytes.

LOC=EXPLICIT
LOC=(EXPLICIT,64)
Indicate that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere in 64-bit storage.

When you specify EXPLICIT on a request for storage from the same virtual page as previously requested storage, you must request it in the same key, subpool, and central storage area as on the previous storage request. For example, if you request virtual storage backed with central storage below 16 megabytes, any subsequent requests for storage from that virtual page must be specified as LOC=(EXPLICIT,24).

,INADDR=*stor addr*
Specifies the desired virtual address for the storage to be obtained. When you specify INADDR, you must specify EXPLICIT on the LOC parameter.

Note:

1. The address specified on INADDR must be on a doubleword boundary.
2. Make sure that the virtual storage address specified on INADDR and the central storage backing specified on the LOC=EXPLICIT parameter are a valid combination. For example, if the address specified on INADDR is for virtual storage above 16 megabytes, specify LOC=EXPLICIT or LOC=(EXPLICIT,ANY). Valid combinations include:
   - Virtual above, central any
   - Virtual any, central any
   - Virtual below, central below
   - Virtual below, central any

,OWNER=HOME
,OWNER=PRIMARY
,OWNER=SECONDARY
,OWNER=SYSTEM
Specifies the entity to which the system will assign ownership of requested CSA, ECSA, SQA, and ESQA storage. The system uses this ownership information to track the use of CSA, ECSA, SQA and ESQA storage. This parameter can have one of the following values:

HOME
The home address space.

PRIMARY
The primary address space.

SECONDARY
The secondary address space.

SYSTEM
The system (the storage is not associated with an address space); specify this value if you expect the requested storage to remain allocated after termination of the job that obtained the storage.

The default value is OWNER=HOME. The system ignores the OWNER parameter unless you specify a CSA, SQA, ECSA, or ESQA subpool on the SP parameter. The OWNER parameter is valid only on the VC, VU, RC, RU, VRC, and VRU types of GETMAIN requests.

Programs that issue the GETMAIN macro with the OWNER parameter can run on any z/OS system.

,CHECKZERO=YES
,CHECKZERO=NO
Specifies whether or not the return code for a successful completion should indicate if the system has cleared the requested storage to zeroes. When CHECKZERO=NO is specified or defaulted, the return code for a successful completion is 0. When CHECKZERO=YES is specified, the return code for a successful completion is X'14' if the system has cleared the requested storage to zeroes, and 0 if the system has not cleared the requested storage to zeroes.

There is no performance cost to specifying CHECKZERO=YES.

Programs that issue the GETMAIN macro with the CHECKZERO parameter can run on any z/OS system. On a down-level system, CHECKZERO will be ignored, and the return code for a successful completion (conditional or unconditional) will be 0.

,RELATED=*value*
Specifies information used to self-document macros by "relating" functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid assembler character string.

ABEND codes

Abend codes the GETMAIN macro might issue are listed below in hexadecimal. For detailed abend code information, see *z/OS MVS System Codes*.

- 104
- 10A
- 178
- 204
- 20A
- 278
- 30A
- 378
- 40A
- 478
- 504
- 604
- 704
- 70A
- 778
- 804
- 80A
- 878
- 90A
- 978
- A0A
- A78
- B04
- B0A
- B78
- D04
- D0A
- D78

Return and reason codes

When the GETMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

Table 23. Return Codes for the GETMAIN Macro

Return Code	Meaning and Action
0	**Meaning**: Successful completion. CHECKZERO=YES was not specified, or the system has not cleared the requested storage to zeroes. **Action**: None.
4	**If you did not specify EXPLICIT on the LOC parameter**: • **Meaning**: Environmental or system error. Virtual storage was not obtained because insufficient storage is available. • **Action**: If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit. If the request is for common (global) storage, your system is probably experiencing a common storage shortage and your request cannot be satisfied until the shortage is corrected. **If you specified EXPLICIT on the LOC parameter**: • **Meaning**: Program error. Virtual storage was not obtained because part of the requested storage area is outside the bounds of the user region. • **Action**: Determine why your program is mistakenly requesting storage outside the user region. If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit.
8	**Meaning**: System error. Virtual storage was not obtained because the system has insufficient central storage to back the request. **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected.
C	**Meaning**: System error. Virtual storage was not obtained because the system cannot page in the page table associated with the storage to be allocated. **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected.
10	**Meaning**: Program error. Virtual storage was not obtained for one of the following reasons: This reason code applies only to GETMAIN requests with LOC=EXPLICIT specified. • Part of the requested area is allocated already. • Virtual storage was already allocated in the same page as this request, but one of the following characteristics of the storage was different: – The subpool – The key – Central storage backing **Action**: Determine why your program is attempting to obtain allocated storage or why your program is attempting to obtain virtual storage with different attributes from the same page of storage. Correct the coding error.
14	**Meaning**: Successful completion. The system has cleared the requested storage to zeroes. This return code occurs only when CHECKZERO=YES is specified. **Action**: None.

Example 1

Obtain 400 bytes of storage from subpool 10. If the storage is available, the address will be returned in register 1 and register 15 will contain 0; if storage is not available, register 15 will contain 4.

```
GETMAIN   RC,LV=400,SP=10
```

Example 2

Obtain 48 bytes of storage from default subpool 0. If the storage is available, the address will be stored in the word at AREAADDR; if the storage is not available, the task will be abnormally terminated.

```
         GETMAIN EU,LV=48,A=AREAADDR
         .
         .
         .
AREAADDR DS      F
```

Example 3

Obtain a minimum of 1024 bytes to a maximum of 4096 bytes of virtual storage from default subpool 0 with virtual and central storage locations either above or below 16 megabytes. If the storage is available, the starting address is to be returned in register 1 and the length of the storage allocated is to be returned in register 0; if the storage is not available, the caller is to be terminated.

```
GETMAIN VRU,LV=(4096,1024),LOC=ANY
```

Example 4

Obtain 248 bytes of storage from subpool 0 using branch entry. To obtain storage from subpool 0, a supervisor state and PSW key 0 caller must specify subpool 240 or 250. If the storage cannot be obtained, the caller is abnormally terminated.

```
GETMAIN EU,LV=248,A=AREAADDR,BRANCH=YES,SP=250.
```

Example 5

Obtain 4096 bytes of storage from CSA subpool 231. Assign the storage area storage key 2. Indicate that the system is to assign the storage to the primary address space. If the storage cannot be obtained, do not abnormally terminate the caller.

```
GETMAIN RC,LV=4096,SP=231,BRANCH=(YES,GLOBAL),BNDRY=PAGE,KEY=2,OWNER=PRIMARY
```

Chapter 20. GQSCAN — Extract information from global resource serialization queue

Description

Use the GQSCAN macro to obtain the status of resources and requestors of those resources. The GQSCAN macro allows you to obtain resource information from the system.

ISGQUERY is the IBM recommended replacement for the GQSCAN service.

The ISGRIB macro allows you to interpret the data that the GQSCAN service routine returns to the user-specified area. The ISGRIB macro maps the resource information block (RIB) and the resource information block extent (RIBE) as shown in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

There are two fields in the RIB that you can use to determine whether any RIBEs were not returned:

- RIBTRIBE contains the total number of RIBEs associated with this RIB
- RIBNRIBE contains the total number of RIBEs returned by GQSCAN with this RIB in the user-specified area indicated by the AREA parameter.

Global resource serialization counts and limits the number of outstanding global resource serialization requests. A global resource serialization request is any ENQ, RESERVE, or GQSCAN that causes an element to be inserted into a queue in the global resource serialization request queue area. See “Limiting global resource serialization requests” in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key. For the SCOPE=GLOBAL and SCOPE=LOCAL parameters, supervisor state.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
Any PASN, any HASN, any SASN	
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

To interpret the data that the GQSCAN service routine returns in the user-specified area, you must include the ISGRIB mapping macro as a DSECT in your program.

Restrictions

Unauthorized callers of GQSCAN need to be authorized through System Authorization Facility (SAF) when Multi-level security (MLS) is active. If the caller is not authorized, the request will fail.

When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=QSCAN must have at least READ authorization to the ISG.QSCANSERVICES.AUTHORIZATION resource in the FACILITY class. You can activate the multilevel security support through the SETROPTS MLACTIVE option in RACF. For general information about defining profiles in the FACILITY class, see *z/OS Security Server RACF Command Language Reference* and *z/OS Security Server RACF Security Administrator's Guide*. For information about multilevel security, see *z/OS Planning for Multilevel Security and the Common Criteria*.

Input register information

Before issuing the GQSCAN macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Register 0 contains a fullword reason code if the return code in register 15 is `X'0A'` or `X'0C'`. Otherwise, register 0 contains the following two halfword values:

- The first (high-order) halfword contains the length of the fixed portion of each RIB returned.
- The second (low-order) halfword contains the length of each RIBE returned or reason code.

1
Contains the number of RIBs that were copied into the area provided

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

In general, the narrower the search parameters (particularly QNAME and RNAME), the less time it takes. Using both a specific QNAME and a specific RNAME gives better performance than using generic prefix.

The use of XSYS=YES (the default) might greatly degrade the performance of the request. Use it only when required.

Polling for ENQ contention through GQSCAN or ISGQUERY is not recommended. See the *z/OS MVS Planning: Global Resource Serialization* and *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about monitoring contention through ENF 51.

When you specify SCOPE=GLOBAL, or SCOPE=LOCAL, the performance of programs that issue ENQ, DEQ, or the RESERVE macro might be temporarily degraded while the GQSCAN service is running.

Syntax

The standard form of the GQSCAN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede GQSCAN.
GQSCAN	
␣	One or more blanks must follow GQSCAN.
AREA=(*area addr*,*area size*)	*area addr*: A-type address or register (2) - (12).
	area size: symbol, decimal digit, or register (2) - (12). **Note:** AREA cannot be specified with QUIT=YES.
,REQLIM=*value*	*value*: symbol, decimal digit, register (2) - (12), or the word MAX.
,REQLIM=MAX	**Default**: REQLIM=MAX
,SCOPE=ALL	**Default**: SCOPE=STEP
,SCOPE=STEP	
,SCOPE=SYSTEM	
,SCOPE=SYSTEMS	
,SCOPE=LOCAL	
,SCOPE=GLOBAL	
,RESERVE=YES	**Default**: All resources requested with RESERVE and all resources requested with ENQ.
,RESERVE=NO	
,RESNAME=(*qname*	*qname addr*: RX-type address or register (2) - (12).
addr[,*rname addr*,	*rname addr*: RX-type address or register (2) - (12).

Syntax	Description
rname length],	*rname length*: decimal digit, or register (2) - (12). **Default**: assembled length of *rname*.
[GENERIC|SPECIFIC],	
qname length)	**Default**: *qname length* of eight.
,SYSNAME=(*sysname addr*	*sysname addr*: RX-type address or register (2) - (12).
[,*asid value*])	*asid value*: symbol, decimal digit, or register (2) - (12). **Note:** Provide *rname addr* only when *qname addr* is used. Code *rname length* if a register is specified for *rname addr.* Code an *asid value* only when the *sysname addr* is used.
,QUIT=YES	**Default**: QUIT=NO
,QUIT=NO	**Note:** QUIT=YES is mutually exclusive with all parameters but TOKEN and MF.
,REQCNT=*value*	*value*: decimal digit or register (2) - (12).
	Default: REQCNT=0
,OWNERCT=*value*,WAITCNT=	
value	*value*: decimal digit or register (2) - (12).
,OWNERCT=*value*	*value*: decimal digit or register (2) - (12).
,WAITCNT=*value*	*value*: decimal digit or register (2) - (12).
,TOKEN=*addr*	*addr*: RX-type address or register (2) - (12).
,XSYS=YES	**Default**: XSYS=YES
,XSYS=NO	**Note:** XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.

Parameters

The parameters are explained as follows:

AREA=(*area addr*,*area size*)
Specifies the location and size of the area where information extracted from the global resource serialization resource queues is to be placed. The minimum size is the amount needed to describe a single resource, which is the length of the fixed portions of the RIB and the maximum size rname rounded up to a fullword value. IBM recommends that you use a minimum of 1024 bytes as the area size.

,REQLIM=*value*
,REQLIM=MAX
Specifies the maximum number of owners and waiters to be returned for each individual resource within the specification of RESNAME, which can be any value in the range 0 to 2^{15}–1. MAX specifies 2^{15}–1 (32767).

,SCOPE=ALL
,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
,SCOPE=LOCAL
,SCOPE=GLOBAL
Specifies that you want information only for resources having the indicated scope. STEP, SYSTEM, or SYSTEMS is the scope specified on the resource request. If you specify SCOPE=ALL (meaning STEP, SYSTEM, and SYSTEMS), the system returns information for all resources the system recognizes that have the specified RESNAME, RESERVE, or SYSNAME characteristics. If you specify SCOPE=LOCAL, information is returned about this system's resources that are not being shared with other systems in the global resource serialization complex. If you specify SCOPE=GLOBAL, information is returned about resources that are being shared with other systems in the global resource serialization complex. Remember that entries in the resource name lists can cause the scope to change.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller's system.

,RESERVE=YES
,RESERVE=NO
If you specify RESERVE=YES, information is only returned for the requestors of the resource, that requested the resource with the RESERVE macro. If, for example, the resource also had requestors with the ENQ macro, the ENQ requestor's information would not be returned for the resource.

RESERVE=NO information is only returned for the requestors of the resource that requested the resource with the ENQ macro. In other words, if the resource also had requestors with the RESERVE macro, the RESERVE requestor's information would not be returned for the resource.

,RESNAME=(*qname addr*[,*rname addr*,*rname length*],[GENERIC|SPECIFIC],*qname length*)
RESNAME identifies an individual resource or group of resources that GQSCAN will examine.

RESNAME with *(rname)* indicates the name of one resource.

The *qname addr* specifies the address of the 8-character major name of the requested resource.

The *rname addr* specifies the virtual storage address of a 1 to 255-byte minor name used with the major name to represent a single resource. Information returned is for a single resource unless you specify SCOPE=ALL, in which case it could be for three resources (STEP, SYSTEM, and SYSTEMS) or SCOPE=LOCAL in which case it could be for two resources (STEP and SYSTEM) if there is a matching name in each of these categories. If the name specified by *rname* is defined by an EQU assembler instruction, the *rname length* must be specified.

The *rname length* specifies the length of the minor name. If you use the register form, specify length in the low-order (rightmost) byte. The length must match the *rname length* specified on ENQ or RESERVE.

GENERIC specifies that the *rname* of the requested resource must match but only for the length specified. For example, an ENQ for SYS1.PROCLIB would match the GQSCAN *rname* specified as SYS1 for an *rname* length of 4.

SPECIFIC specifies that the *rname* of the requested resource must exactly match the GQSCAN *rname*.

Note: GENERIC and SPECIFIC are mutually exclusive.

The *qname* length specifies the number of characters in a resource *qname* that must match the GQSCAN *qname* specified by RESNAME. You must specify a *qname* length to request a GQSCAN for a generic *qname*. For example, an ENQ with a *qname* of SYSDSN would match a GQSCAN specifying GENERIC with a *qname* of SYSD and *qname* length of 4. Specify zero for the *qname* length (with any *qname*) to request a generic GQSCAN matching any resource *qname*. If you do not specify a *qname* length, GQSCAN uses the default of 8.

,SYSNAME=(*sysname addr*[,*asid value*])
Specify SYSNAME to tell GQSCAN to return information for resources requested by tasks running on the MVS system specified in an 8-byte field pointed to by the address in *sysname address* and the *asid value*, a 4-byte address space identifier, right justified. Valid SYSNAMEs are specified in the IEASYS*xx* parmlib member.

Information returned includes only those resources whose *sysname addr* and *asid value* match the ones specified. SYSNAME=0 or SYSNAME=(0,*asid value*), specifies that the system name is that of the system on which GQSCAN is issued. The system issues return code X'0A' with a reason code of X'0C', if SYSNAME≠0 or SYSNAME≠(0,*asid value*) is specified with XSYS=NO.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller's system.

,QUIT=YES
,QUIT=NO
QUIT=NO indicates that you do not want to end the current global resource serialization queue scan. QUIT=YES tells GQSCAN to stop processing the current global resource serialization queue scan and release the storage allocated to accumulate the information specified in the token.

If you specify QUIT=YES, you *must* specify the TOKEN parameter. If you specify QUIT=YES without the TOKEN parameter, the system issues abend X'09A'.

If you specify QUIT=YES **without** the TOKEN parameter, the system issues return code X'0A' with a reason code of X'34'. Specifying QUIT=YES with TOKEN=0 will result in the system issuing return code X'0A' with a reason code of X'2C'.

If you specify QUIT=YES with a token that was previously obtained through GQSCAN with SCOPE=LOCAL or SCOPE=GLOBAL, your program must be in supervisor state when it issues GQSCAN with QUIT=YES.

If you specify QUIT=YES with XSYS=NO, the system issues return code X'0A' with a reason code of X'0C'.

,REQCNT=*rcount*
Specifies that you want GQSCAN to return resource information only when the total number of requesters (owners plus waiters) for an individual resource is greater than or equal to *rcount*, which can be any value in the range 0 to $2^{31}-1$.

,OWNERCT=*ocount*
Specifies that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to *ocount*, which can be any value in the range 0 to $2^{31}-1$.

,WAITCNT=*wcount*
Specifies that you want GQSCAN to return resource information only when the total number of waiters for an individual resource is greater than or equal to *wcount*, which can be any value in the range 0 to $2^{31}-1$.

OWNERCT=*ocount*,WAITCNT=*wcount*
Specifies that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to *ocount* or when the total number of waiters for an individual resource is greater than or equal to *wcount*.

,TOKEN=*addr*
Specifies the address of a fullword of storage that the GQSCAN service routine can use to provide you with any remaining information in subsequent invocations. If the token value is zero, the scan starts at the beginning of the resource queue. If the token value is not zero, the scan resumes at the point specified on TOKEN. Specify the same token value that GQSCAN returned on its previous invocation to continue where processing left off on the previous invocation.

When providing a non-zero token value, you must specify the same scope that you specified on the GQSCAN request that returned the token.

,XSYS=YES
,XSYS=NO
Specifies whether GQSCAN should be propagated across systems in the global resource serialization complex, to gather complex-wide information. This parameter is ignored in a global resource serialization ring complex, and for requests that only gather local data.

Specify XSYS=YES if the program requires complex-wide global resource serialization information. The caller might be suspended while the information is being gathered. Do *not* specify or default to XSYS=YES if this condition cannot be tolerated.

Specify XSYS=NO if the program will accept global resource serialization information from this system only. The RIBE data will contain information about requestors from other other systems in the complex only if that information is already available on the GQSCAN caller's system. Otherwise, RIBE data will be provided only for requests from the GQSCAN caller's system, and the counts in the RIB will reflect only those requests. This request is always handled without placing the caller's dispatchable unit into a wait.

ABEND codes

See *z/OS MVS System Codes* for more information about the abend codes.

Return and reason codes

When GQSCAN returns control, register 15 contains one of the following return codes:

Table 24. Return codes for the GQSCAN macro

Hexadecimal return code	Meaning and action
00	**Meaning:** Queue scan processing is complete. Data is now in the area you specified. There is no more data to return. **Action:** Process the data.
04	**Meaning:** **Action:** **Meaning**: Queue scan processing is complete. No resources matched your request. **Action**: Do not try to process any data; none exists.
08	**Meaning:** The area you specified was filled before queue scan processing completed. **Action:** If you specified TOKEN, process the information in the area and issue GQSCAN again, specifying the TOKEN returned to you. If you did not specify TOKEN, specify a larger area or specify a TOKEN.

Table 24. Return codes for the GQSCAN macro (continued)

Hexadecimal return code	Meaning and action
0A	**Meaning:** The information you specified to GQSCAN is not valid. **Action:** Take the action indicated by the following hexadecimal reason code found in register 0. **Reason code** **Meaning** **04** The caller attempted to use GQSCAN before the global resource serialization (GRS) address space was active. **08** The size of the reply area, specified by the AREA parameter, is too small to contain a resource information block (RIB) of maximum size. **0C** You specified mutually exclusive arguments (RESERVE=YES, RESERVE=NO, RESNAME=, SYSNAME=, or XSYS=NO) to GQSCAN. **10** The caller was holding a local lock other than the GRS local lock when GQSCAN was invoked. **14** One of the following conditions, in reference to the RESNAME parameter, was detected by GQSCAN: • The *qname length* was specified with a value greater than eight. • The *qname length* value was specified without the *qname addr* value. • The SPECIFIC parameter was specified with a *rname length* value of zero. • The *rname* or *rname length* was specified without the *qname addr* value. **18** The *asid value*, for the SYSNAME parameter was specified without the *sysname addr* value. **1C** The REQCNT parameter was specified with either the OWNERCNT or WAITCNT parameters. **20** The combination of values specified on the SCOPE parameter is not valid. **28** An element in GQSCAN's input parameter list was not in the caller's storage protect key. **2C** An invalid token was specified to GQSCAN. **30** The GQSCAN caller is not authorized to use the restricted interface (SCOPE=LOCAL or GLOBAL). **34** QUIT=YES was specified without the TOKEN parameter. **38** The caller held a CMS lock other than CMSEQDQ when GQSCAN was invoked. **3C** The caller held a lock that violated the environmental restrictions of a service required by GQSCAN. **40** The caller invoked GQSCAN in the service request block (SRB) mode. **44** The value specified for the REQLIM parameter was not valid. **48** The value specified for the REQCNT parameter was not valid. **4C** The value specified for the OWNERCT parameter was not valid. **50** The value specified for the WAITCNT parameter was not valid. **54** The GQSCAN token (TOKEN) is expired. **58** SETROPTS MLACTIVE is in effect, and the program is not authorized to issue GQSCAN. Ensure the program is running authorized, or is associated with a userid with at least READ access to the best fit FACILITY class resource profile of the form ISG.QSCANSERVICES.AUTHORIZATION and that the FACILITY class is SETROPTS RACLISTed.

Table 24. Return codes for the GQSCAN macro (continued)	
Hexadecimal return code	**Meaning and action**
0C	**Meaning:** System error. Queue scan encountered an abnormal situation while processing. The information in your area is not meaningful. The reason code in register 0 contains one of the following: **Reason code** **Meaning** **00** GQSCAN has sustained an unrecoverable error. **04** The GQSCAN caller attempted to resume a scan that was started when the global resource serialization complex, which is now in star mode, was in ring mode. **08** The GQSCAN service is not able to obtain storage to satisfy the request. **0C** Sysplex processing of a SYSTEMS or GLOBAL request failed. **10** The GQSCAN service failed because the complex was migrating from a ring to a star configuration. **14** The GQSCAN service failed because inconsistent data was returned from one or more systems. **Action:** Do not try to process any data; none exists. Retry the request one or more times.
10	**Meaning:** Program error. An incorrect SYSNAME was specified as input to queue scan. The information in your area is not meaningful. **Action:** Specify a valid SYSNAME on the call to GQSCAN.
14	**Meaning:** Environmental error. The area you specified was filled before queue scan processing completed. Your request specified TOKEN, but the limit for the number of concurrent resource requests (ENQ, RESERVE, or GQSCAN) has been reached. The information in your area is valid but incomplete. The scan cannot be resumed. **Action:** Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.

GQSCAN - List form

The list form of the GQSCAN macro is used to construct a non-executable parameter list. This parameter list, or a copy of it for reentrant programs, can be referred to by the execute form of the GQSCAN macro.

The list form of the GQSCAN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GQSCAN.
GQSCAN	
ƀ	One or more blanks must follow GQSCAN.
AREA=*(area addr, area size)*	*area addr*: A-type address.
	area size: symbol, decimal digit.
	Note: 1. This parameter cannot be specified with QUIT=YES. 2. AREA is required on either the list or the execute form of the macro.

Syntax	Description
,REQLIM=*value*	*value*: symbol, decimal digit or the word MAX.
,REQLIM=MAX	**Default**: REQLIM=MAX
,SCOPE=ALL	**Default**: SCOPE=STEP
,SCOPE=STEP	
,SCOPE=SYSTEM	
,SCOPE=SYSTEMS	
,RESERVE=YES	**Default**: All resources requested with RESERVE and all
,RESERVE=NO	resources requested with ENQ.
,RESNAME=*(qname*	*qname addr*: A-type address.
addr [,rname addr,	*rname addr*: A-type address.
rname length],	*rname length*: decimal digit.
[GENERIC|SPECIFIC],	**Default**: assembled length of rname.
qname length)	**Default**: *qname length* of eight.
,SYSNAME=*(sysname addr*	*sysname addr*: A-type address.
[,asid value])	*asid value*: symbol, decimal digit.
	Note: *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES	**Default**: QUIT=NO
,QUIT=NO	**Note:** Only TOKEN and MF=L can be specified with QUIT=YES.
,REQCNT=*value*	*value*: decimal digit. **Default**: REQCNT=0
,OWNERCT=*value*,WAITCNT=	
value	*value*: decimal digit.
,OWNERCT=*value*	*value*: decimal digit.
,WAITCNT=*value*	*value*: decimal digit.
,TOKEN=*addr*	*addr*: RX-type address.

Syntax	Description
,XSYS=YES	**Default**: XSYS=YES
,XSYS=NO	**Note:** XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.
,MF=L	

Parameters

The parameters are explained under the standard form of the GQSCAN macro with the following exception:

,MF=L
Specifies the list form of the GQSCAN macro.

GQSCAN - Execute form

The execute form of the GQSCAN macro can refer to and modify a remote parameter list built by the list form of the macro. There are no defaults for any of the parameters in the execute form of the macro.

The execute form of the GQSCAN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GQSCAN.
GQSCAN	
ƀ	One or more blanks must follow GQSCAN.
AREA=*(area addr,area size)*	*area addr*: RX-type address or register (2) - (12).
	area size: symbol, decimal digit, or register (2) - (12).
	Note: 1. AREA cannot be specified with QUIT=YES. 2. AREA is required on either the list or the execute form of the macro.
,REQLIM=*value*	*value*: symbol, decimal digit, register (2) - (12), or the word MAX.
,REQLIM=MAX	

Syntax	Description
,SCOPE=STEP	**Note:** SCOPE=LOCAL and SCOPE=GLOBAL cannot be coded on the list form of this macro.
,SCOPE=ALL	
,SCOPE=SYSTEM	
,SCOPE=SYSTEMS	
,SCOPE=LOCAL	
,SCOPE=GLOBAL	
,RESERVE=YES	
,RESERVE=NO	
,RESNAME=*(qname*	*qname addr*: RX-type address or register (2) - (12).
addr[,rname addr,	*rname addr*: RX-type address or register (2) - (12).
rname length],	*rname length*: decimal digit, register (2) - (12). **Default**: assembled length of rname.
[GENERIC|SPECIFIC],	
qname length)	
,SYSNAME=*(sysname addr*	*sysname addr*: RX-type address or register (2) - (12).
[,asid value]	*asid value*: symbol, decimal digit, or register (2) - (12).
	Note: *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr.* An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES	**Default**: QUIT=NO
,QUIT=NO	**Note:** Only TOKEN and MF=(E, *parm list addr*) can be specified with QUIT=YES.
,REQCNT=*value*	*value*: decimal digit or register (2) - (12).
	Default: REQCNT=0
,OWNERCT=*value*,WAITCNT= *value*	*value*: decimal digit.
,OWNERCT=*value*	*value*: decimal digit.
,WAITCNT=*value*	*value*: decimal digit.

Syntax	Description
,TOKEN=*addr*	*addr*: RX-type address of register (2) - (12).
,XSYS=YES	**Default**: XSYS=YES
,XSYS=NO	**Note:** XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained under the standard form of the GQSCAN macro with the following exception:

,MF=(E,*list addr)*
Specifies the execute form of the GQSCAN macro.

list addr specifies the area that the system uses to contain the parameters.

Chapter 21. GTRACE — GTF trace recording

Description

Use the GTRACE macro to record system or application errors through the generalized trace facility (GTF). The GTRACE macro provides three separate functions, depending on the keyword specified:

- GTRACE TEST indicates whether the operator requested a specific user event.
- GTRACE QUERY indicates how much data GTF can store when a program issues GTRACE DATA.
- GTRACE DATA generates GTF trace records for specific events.

Refer to *z/OS MVS Diagnosis: Tools and Service Aids* and *z/OS Problem Management* for information about using GTF.

The following description of the GTRACE macro is divided into three sections, one for each function of the macro. The TEST and QUERY functions have only one form each, while the DATA function has standard, list, and execute forms.

GTRACE TEST

The TEST function of the GTRACE macro indicates whether the operator requested a particular user event in response to the USRP option. The system returns the test result as a return code in register 15.

By issuing GTRACE TEST and checking the return code, you can determine whether you need to subsequently issue GTRACE DATA to write the record. If the return code indicates that tracing has been requested by USRP for the specified user event, then issue GTRACE DATA.

Issuing GTRACE TEST before issuing GTRACE DATA is not necessary but you might find it useful to do so if the processing of your code can benefit from learning whether processing is active for the record type you want to record to the generalized trace facility (GTF) before requesting to do that recording.

When the operator requests GTF prompting for specific event identifiers with the USRP option and your program issues GTRACE DATA, the system records the user trace record only when the event identifier specified on GTRACE DATA was also requested with the USRP option. However, the TEST function is still supported for compatibility with existing programs.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space and all data must reside in primary address space.

Programming requirements

- Include the CVT and the MCHEAD mapping macros.
- When you code the CVT mapping macro, you must not specify PREFIX=YES.

Restrictions

None.

Input register information

Before issuing the GTRACE TEST macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Unchanged

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

Performance implications

None.

Syntax

The TEST function of the GTRACE macro is coded as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GTRACE.
GTRACE	
ƀ	One or more blanks must follow GTRACE.
TEST=YES	

Syntax	Description
,ID=*id*	*id*: Symbol, decimal digit, or hexadecimal number.

Parameters

The parameters are explained as follows:

TEST=YES
Specifies the test function of the GTRACE macro.

,ID=*id*
Specifies the event ID for the user event that is to be tested. Decimal event IDs 0 through 1023 (X'3FF') are available for user events. You can specify the ID in decimal or in hexadecimal. Use the expression X'*id*' to specify a hexadecimal number.

ABEND codes

None.

Return codes

When GTRACE TEST macro returns control to your program, GPR 15 contains a return code.

Table 25. Return Codes for the GTRACE TEST Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Tracing has not been requested by USRP for the specified user event. **Action**: Do not issue a GTRACE DATA request to create your trace record for the specified user event ID.
04	**Meaning**: Tracing has been requested by USRP for the specified user event. **Action**: You may issue a GTRACE DATA request to create your trace record for the specified user event ID.

GTRACE QUERY

The QUERY function of the GTRACE macro determines how much data GTF will accept, and returns the value in the variable or register specified with the MAXLNG parameter. This function is useful when your program must run on different levels of MVS that accept different amounts of trace data in GTRACE DATA.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O or external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the GTRACE QUERY macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-14
Unchanged

15
Zero

Performance implications

None.

Syntax

The QUERY function of the GTRACE macro is coded as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GTRACE.
GTRACE	
ƀ	One or more blanks must follow GTRACE.
QUERY	
,MAXLNG=*addr*	*addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

QUERY
Specifies the query function of the GTRACE macro.

,MAXLNG=*addr*
Specifies the address in which the maximum amount of GTF data is returned.

ABEND codes

None.

Return codes

The return code from GTRACE QUERY is always zero.

GTRACE DATA

The DATA function of the GTRACE macro records system or problem program data in the GTF trace buffers. GTRACE DATA can trace up to 8192 bytes of data.

Data is written only if you requested the event qualifier (through the USRP option) when you started GTF. Therefore, you can issue the GTRACE DATA without issuing a GTRACE TEST.

In earlier releases, GTRACE DATA writes the record to the GTF data set even if the record's event ID (EID) is excluded from a USRP list in the GTF trace options. Therefore, you need to issue a GTRACE TEST before you issue GTRACE DATA to determine if data is to be collected for the event qualifier.

Environment

The requirements for the caller are:

Environmental factor	**Requirement**
Minimum authorization:	Problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN; all data and parameters must reside in the home address space.
AMODE:	24- or 31- or 64-bit. The caller must be in 31-bit mode for GTRACE to record data above 16 megabytes.
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the GTRACE DATA macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Unchanged

1
Used as a work register by the system

2-14
Unchanged

15
Return code

Performance implications

None.

Syntax

The standard form of the DATA function of the GTRACE macro is coded as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GTRACE.
GTRACE	
ƀ	One or more blanks must follow GTRACE.
DATA=*addr*	*addr*: RX address or register (2) - (12).
DATA64=*addr*	
,LNG=*nbr*	*nbr*: Symbol, decimal number, hexadecimal number, or register (2) - (12).
,ID=*id*	*id*: Symbol, decimal number, or hexadecimal number.
,FID=*fidname*	*fidname*: Symbol, decimal number, hexadecimal number, or register (2) - (12).
,PAGEIN=NO	**Default**: PAGEIN=NO
,PAGEIN=YES	

Syntax	Description

Parameters

The parameters are explained as follows:

DATA=*addr*
DATA64=*addr*
Specifies the virtual storage address of the data that is to be recorded.

Note: DATA64 can be specified only when running in 64-bit address mode (AMODE).

,LNG=*nbr*
Specifies the number of data bytes (1 through 8192) to be recorded from the address specified by the DATA parameter. You can specify the number in decimal or in hexadecimal. If the number is hexadecimal, use the expression X'*nbr*' to specify the number.

Note: When you specify LNG, the trace record contains the number of bytes that you specify plus 12 bytes, which is the size of the trace record header. The header consists of a 4-byte ASCB address followed by an 8-byte jobname. Thus, if you specify LNG=8192, the trace record has 8204 (8192+12) bytes.

,ID=*id*
Specifies the event ID that is to be recorded with the data bytes. Decimal event ids 0 through 1023 (X'3FF') are available for user events. You can specify the ID in decimal or in hexadecimal. Use the expression X'*id*' to specify a hexadecimal number.

,FID=*fidname*
Specifies the format appendage that controls the formatting of this record. Formatting occurs when the trace output is processed by GTF trace. The format appendage name is formed by appending the 2-digit FID value to the names AMDUSR, HMDUSR, and IMDUSR. Assign FID values as follows:

X'00'
The record is to be dumped in hexadecimal.

X'01' to X'50'
The record contains user format identifiers.

Note: If you code FID without any *fidname*, or if you omit the FID parameter, the system supplies a default *fidname* of zero.

,PAGEIN=NO
,PAGEIN=YES
Specifies that paged-out user data is to be processed (YES) or not to be processed (NO). To ensure that all user data is traced, specify YES.

ABEND codes

None.

Return codes

When GTRACE DATA macro returns control to your program, GPR 15 contains a return code.

Table 26. Return Codes for the GTRACE DATA Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: GTF is active. The data was recorded in GTF trace buffers. **Action**: None.

Table 26. Return Codes for the GTRACE DATA Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
04	**Meaning**: GTF is not active or not active for this particular event ID. No data was recorded. **Action**: None.
08	**Meaning**: Program error. The value of the LNG keyword is not valid. It must be a number from 1 through 8192. No data was recorded. **Action**: Reissue the macro, specifying a valid amount of trace data to be recorded.
0C	**Meaning**: Program error. The value of the DATA keyword is not valid. It does not represent an area of storage that the calling program can refer to. No data was recorded. **Action**: Correct the problem and reissue the macro.
10	**Meaning**: Program error. The value of the FID keyword is not valid. It must be a number from X'0' through X'FF'. No data was recorded. **Action**: Correct the problem and reissue the macro.
18	**Meaning**: Environmental condition. All GTF buffers are full. No data was recorded. **Action**: None.
1C	**Meaning**: Program error. The address of the parameter list for GTF is not valid. The parameter list is not in storage that the caller can refer to, or its format is not valid. No data was recorded. **Action**: Correct the problem and reissue the macro.
20	**Meaning**: Program error. Some of the data to be recorded was paged out. No data was recorded. This return code is not valid with PAGEIN=YES. **Action**: Page-fix the storage containing the data to be recorded or modify the macro invocation to specify the PAGEIN=YES option.

Example

Use GTRACE to record 200 bytes of user data plus 12 bytes for the trace record header. The user data is found at symbolic address AREA. Use an event identifier of 37. Use the formatting appendage named IMDUSR40 to control the formatting.

```
GTRACE DATA=AREA,LNG=200,ID=37,FID=X'40'
```

GTRACE DATA - List form

Use the list form of the GTRACE DATA macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form of the macro uses to store the parameters.

The list form of the GTRACE parameter list must reside below the bar.

Syntax

The list form of the DATA function of the GTRACE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GTRACE.

Syntax	Description
GTRACE	
ƀ	One or more blanks must follow GTRACE.
DATA=*addr*	*addr*: A-type address or register (2) - (12).
DATA64=*addr*	
,LNG=*nbr*	*nbr*: Symbol, decimal number, hexadecimal number, or register (2) - (12).
,FID=*fidname*	*fidname*: Symbol, decimal number, hexadecimal number, or register (2) - (12).
,MF=L	

Parameters

The parameters are described under the standard form of the GTRACE DATA macro, with the following exception:

,MF=L
Specifies the list form of the GTRACE DATA macro.

GTRACE DATA - Execute form

Use the execute form of the GTRACE DATA macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the DATA function of the GTRACE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GTRACE.
GTRACE	
ƀ	One or more blanks must follow GTRACE.

Syntax	Description
DATA=*addr*	*addr*: RX address or register (2) - (12).
DATA64=*addr*	
,LNG=*nbr*	*nbr*: Symbol, decimal number, hexadecimal number, or register (2) - (12).
,ID=*id*	*id*: Symbol, decimal number, or hexadecimal number.
,FID=*fidname*	*fidname*: Symbol, decimal number, hexadecimal number, or register (2) - (12). **Note:** If you omit the FID parameter on the execute form of GTRACE, the FID value defaults to zero. This default overlays the FID value that you specify on the list form of GTRACE. If you want the system to obtain the FID value from the remote problem-program parameter list, then you must specify the FID parameter as a null value by coding FID= without any *fidname*.
,PAGEIN=NO	**Default**: PAGEIN=NO
,PAGEIN=YES	
,MF=(E,*parm list addr*)	*parm list addr*: A-type address or register (2) - (12).

Parameters

The parameters are described under the standard form of the GTRACE DATA macro, with the following exception:

,MF=(E,*parm list addr*)
Specifies the execute form of the GTRACE DATA macro using a remote problem-program parameter list.

Chapter 22. HISMT — HIS multithreading service

Description

HISMT provides an interface to retrieve multithreading metrics at different granularity levels between the caller's current and previous HISMT invocations.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state. PSW key 0.
Dispatchable unit mode:	Task or SRB mode.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31-bit.
ASC mode:	Primary.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	Locks may be held.
Control parameters:	Control parameters and the save area must be addressable from the primary address space and must be in DREF/fixed storage.

Programming requirements

There must be at least 1 second between HISMT invocations for the same interval area.

The service can be invoked in any multithreading mode, including a multithreading mode of 1. The multithreading mode of a processor class is the number of active threads per core for a processor class. The multithreading mode for each processor class can be set via the IEAOPT*xx* parmlib member.

The caller must include the HISYMT macro to get a mapping of the interval area (INTVAREA) and constants for the return and reason codes.

HISMT exploiters may register as a listener for ENF 20 (System Information) and if a primary or secondary CPU speeds have changed, invoke HISMT to end the HISMT interval. The current CPU speeds are available in SI22V1PrimaryCPUSpeed and SI22V1SecondaryCPUSpeed fields of the STSI (see macro CSRSIIDF). When the core speed changes during the HISMT interval, HISMT will return a warning reason code HisMT_kRsnWarn_ConfigChanged to inform users of questionable HISMT metrics due to the core state changing during the interval. See field HisMT_Hdr_Flags in the interval area header for what has changed.

The multithreading metrics requested through the HISMT service must be the same on each call for the same interval area (INTVAREA). The size of the interval area (INTVAREALEN) depends on the multithreading metrics requested, machine, and configuration. To allocate a sufficiently large interval area, you must adhere to the following protocol:

1. Obtaining sufficient storage:
   a. As part of initialization, invoke the HISMT service with the requested metrics and pass an interval area that is the minimum interval area length, HisMT_Hdr_kLength. (See INTVAREA parameter for requirements on the first call for a new interval area)
   b. The service will likely complete with a return code HisMT_kRetWarn and reason code HisMT_kRsnWarn_IntvAreaSmall.

c. Using the HisMT_Hdr_LengthRequired field in the interval area returned by the HISMT service, allocate a new interval area that is at least HisMT_Hdr_LengthRequired bytes long.

2. Making the first HISMT call with the new interval area:

   a. Invoke the HISMT service with the same requested multithreading metrics passing the newly allocated interval area that is at least HisMT_Hdr_LengthRequired bytes long. (See INTVAREA parameter for requirements on the first call for a new interval area)

   b. The service will likely complete with a return code HisMT_kRetOk. The first HISMT invocation for a new interval area marks the start of the first HISMT interval. The requested HISMT metrics will not be returned in the new interval area for the first call. Each metric will contain HisMT_Entry_kNoData since the system is unable to calculate the metrics on the very first call. (See Step 3 of the protocol below for instructions on getting the requested HISMT metrics in the interval area)

3. Making a subsequent HISMT call with the previous interval area:

   a. Invoke the HISMT service with the same requested multithreading metrics passing the interval area from the previous invocation, unchanged.

   b. The service will likely complete with a return code HisMT_kRetOk. This invocation marks the end of the current HISMT interval and the start of the next HISMT interval. The requested HISMT metrics will be returned in the interval area for the current interval (the time between the previous HISMT invocation and this HISMT invocation). The HISMT interval area has descriptors for each requested metric that contain information on how to locate and process each array of metric values. For all requested metrics, these descriptors contain the offset to the first element in the array of metric values, the number of elements in the array, and the size each value in the metric array. See macro HISYMT for more information.

For example, with multithreading, to calculate how much single thread capacity an MT=2 core of a particular processor class can deliver for the current workload over an interval for this workload, calculate:

```
interval × Procclass Max Capacity Factor / HisMT_Entry_kMetricFactor
```

(Assembler programs can shift right by HisMT_Entry_kMetricShift bits for the division.)

Input register information

Before issuing the HISMT macro, the caller must ensure that the following general purpose register (GPR) contains the specified information.

GPR
Contents

13
The address of a 144-byte F4SA format save area in the primary address space

Before issuing the HISMT macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HISMT macro, the caller does not have to place any information into any access register (AR).

Output register information

When control returns to the caller, the GPRs contain:

GPR
Contents

0
Reason code if GPR15 is not 0

1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The HISMT macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede HISMT.
HISMT	
␣	One or more blanks must follow HISMT.
INTVAREA=*xintvarea*	
,INTVAREALEN=*xintvarealen*	
,AVGTDCLASS=NO	**Default:** AVGTDCLASS=NO
,AVGTDCLASS=YES	
,CAPCLASS=NO	**Default:** CAPCLASS=NO
,CAPCLASS=YES	

Syntax	Description
,COREBUSYTIME=NO	**Default:** COREBUSYTIME=NO
,COREBUSYTIME=YES	
,MAXCAPCLASS=NO	**Default:** MAXCAPCLASS=NO
,MAXCAPCLASS=YES	
,PRODCLASS=NO	**Default:** PRODCLASS=NO
,PRODCLASS=YES	
,PRODCORE=NO	**Default:** PRODCORE=NO
,PRODCORE=YES	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15),
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (0) or (2) - (12), (00), (GPR
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the HISMT macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

INTVAREA=*xintvarea*
is the name (RS-type), or address in register (2) - (12), of a required character input/output that contains the interval area. It must be in DREF or fixed storage. Note that the interval area must start

and end on a double word boundary. The size of the interval area depends on the MT metrics requested, machine, and configuration. If this is the first HISMT request for a new interval area, the first byte of the interval area must contain binary zeroes. If this is a subsequent HISMT request, pass the interval area that was returned by the previous HISMT invocation, unchanged. Macro HISYMT contains the mapping of the interval area. The minimum amount of storage required for the request is HisMT_Hdr_kLength. See "Programming requirements" on page 251 for the protocol for passing an interval area that is long enough to accommodate the request.

,INTVAREALEN=*xintvarealen*
is the name (RS-type), or address in register (2) - (12), of a required fullword input that contains the length of the provided interval area. The minimum INTVAREALEN required for the request is HisMT_Hdr_kLength. Note that the interval area length must be a multiple of 8 bytes.

,AVGTDCLASS=NO
,AVGTDCLASS=YES
Indicates whether Average Thread Density must be returned by processor class granularity and can be located using HisMT_Hdr_AvgTDClass_Desc. Average Thread Density is the average number of active threads for active cores (dispatched to physical hardware) within a processor class. If the system cannot calculate this value, the value will contain HisMT_Entry_kNoData. For example, the AVGTDCLASS will contain HisMt_Entry_kNoData when there are no cores defined in a processor class, when all cores are offline in a processor class, when all cores in a processor class change ONLINE/OFFLINE status during an HISMT interval or when no cores in a processor class were dispatched to physical hardware.

The default is AVGTDCLASS=NO

.

,AVGTDCLASS=NO
Average thread density is not needed for any processor class.

,AVGTDCLASS=YES
Average thread density is needed for each processor class.

,CAPCLASS=NO
,CAPCLASS=YES
Indicates whether multithreading capacity factor metrics must be returned by processor class granularity and can be located using HisMT_Hdr_CapClass_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMT_Entry_kNoData. For example, the CAPCLASS metric will contain HisMt_Entry_kNoData when there are no cores defined in a processor class.

The multithreading capacity factor is a metric that represents a ratio of how much work was accomplished at the current multithreading mode to the amount of work (for the same workload) that could have been accomplished while running with a multithreading mode of 1. For a multithreading mode of 1, a processor class will achieve a capacity factor ratio of 1.0 (100%) because whenever cores are dispatched to physical hardware, they are executing as much work as possible. For example, if the multithreading mode was greater than 1 (100%) and the MT capacity factor was 1.3 (130%), it means for the workload running, the cores were able to accomplish 1.3 times (or 130%) the work than the processor class running with a multithreading mode of 1 would have accomplished for the same workload.

The default is CAPCLASS=NO

,CAPCLASS=NO
Multithreading capacity factor metric is not needed for any processor class.

,CAPCLASS=YES
Multithreading capacity factor metric is needed for each processor class.

,COREBUSYTIME=NO
,COREBUSYTIME=YES
Indicates whether core busy time must be returned by core granularity and can be located using HisMT_Hdr_CoreBusyTime_Desc.

Core busy time is the amount of time (in milliseconds) a logical core was dispatched to a physical core over some interval. If the system cannot calculate this value, the value will contain HisMT_Entry_kNoData. For example, the COREBUSYTIME will contain HisMt_Entry_kNoData for an undefined core.

The default is COREBUSYTIME=NO

,COREBUSYTIME=NO
Core busy time is not needed for any core.

,COREBUSYTIME=YES
Core busy time is needed for each core.

,MAXCAPCLASS=NO
,MAXCAPCLASS=YES
Indicates whether multithreading maximum capacity factor metrics must be returned by processor class granularity and can be located using HisMT_Hdr_MaxCapClass_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMT_Entry_kNoData. For example, the MAXCAPCLASS metric will contain HisMt_Entry_kNoData when there are no cores defined in a processor class.

Multithreading max capacity factor is a metric that represents a ratio of the maximum amount of work that can be accomplished using all active threads at the current multithreading mode to the amount of work (for the same workload) that would have been accomplished while running with a multithreading mode of 1. For a multithreading mode of 1, a processor class will achieve a max capacity factor ratio of 1.0 (100%) because whenever cores are dispatched to physical hardware, they are executing the maximum amount of work as possible. For example, if the multithreading mode is greater than 1 and the multithreading max capacity factor was 1.4 (140%), it means that if the processor class was able to achieve a productivity ratio of 1.0, then the workload running would be able to accomplish 1.4 times (140%) the work a processor class running with a multithreading mode of 1 would have accomplished for the same workload.

The default is MAXCAPCLASS=NO

,MAXCAPCLASS=NO
Multithreading max capacity factor metric is not needed for any processor class.

,MAXCAPCLASS=YES
Multithreading max capacity factor metric is needed for each processor class.

,PRODCLASS=NO
,PRODCLASS=YES
Indicates whether multithreading productivity metrics must be returned by processor class granularity and can be located using HisMT_Hdr_ProdClass_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMT_Entry_kNoData. For example, the PRODCLASS metric will contain HisMt_Entry_kNoData when there are no cores defined in a processor class.

Multithreading productivity is a metric that represents a ratio of how much work was accomplished to the maximum amount of work that could have been accomplished. For a multithreading mode of 1, a processor class will achieve a productivity ratio of 1.0 (100%) because whenever cores are dispatched to physical hardware, they are executing as much work as possible. For example, if the multithreading mode is greater than 1 and the productivity ratio is 0.93 (93%), it means the active threads on all cores accomplished 93% of the work that could have been accomplished while dispatched to physical hardware. Typically, when the multithreading productivity is less than 1.0 (<100%), it is because there were times when the cores were dispatched to physical hardware and one or more threads on those cores were in a wait state because they had no work to run.

The default is PRODCLASS=NO

,PRODCLASS=NO
Multithreading productivity metric is not needed for any processor class.

,PRODCLASS=YES
Multithreading productivity metric is needed for each processor class.

,PRODCORE=NO
,PRODCORE=YES
Indicates whether multithreading productivity metrics must be returned by core granularity and can be located using HisMT_Hdr_ProdCore_Desc. Each metric is calculated for the current multithreading mode while the cores are dispatched to physical hardware. If the system cannot calculate this metric, the metric will contain HisMT_Entry_kNoData. For example, the PRODCORE metric will contain HisMt_Entry_kNoData for an undefined or offline core.

See the PRODCLASS keyword for information about multithreading productivity.

The default is PRODCORE=NO

,PRODCORE=NO
Multithreading productivity metric is not needed for any core.

,PRODCORE=YES
Multithreading productivity metric is needed for each core.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX

- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the HISMT macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HISYSERV provides equate symbols for the return and reason codes. Note carefully that bits 0 - 15 of the reason code may contain component diagnostic data and must not be assumed to be 0.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the *xxxx* value.

Table 27. Return and reason codes for the HISMT macro

Return code	Reason code	Equate symbol, meaning, and action
0	–	**Equate Symbol**: HisMT_kRetOk **Meaning**: HISMT request successful. **Action**: Processing continues.
4	–	**Equate Symbol**: HisMT_kRetWarn **Meaning**: Warning **Action**: Refer to the action provided with the specific reason code.
4	*xxxx*0401	**Equate Symbol**: HisMT_kRsnWarn_IntvAreaSmall **Meaning**: The interval area provided was large enough to hold the minimum amount of data required for the request, but not large enough to hold all of the data requested. **Action**: Obtain a larger interval area using the HisMT_Hdr_LengthRequired field returned in the request's INTVAREA. Then call the service with the newly allocated interval area. (See "Programming requirements" on page 251 for the protocol for passing a new interval area that is at least HisMT_Hdr_LengthRequired bytes long)
4	*xxxx*0402	**Equate Symbol**: HisMT_kRsnWarn_ConfigChanged **Meaning**: The system configuration was changed during the interval between the last HISMT call and the current HISMT call. The metric values returned in the interval area is questionable since the configuration was not consistent during the interval. See field HisMT_Hdr_Flags in the interval area header for what has changed. **Action**: The metric values returned from this call can be ignored.
8	–	**Equate Symbol**: HisMT_kRetUser **Meaning**: HISMT request failed due to a user error. **Action**: Refer to the action provided with the specific reason code
8	*xxxx*0801	**Equate Symbol**: HisMT_kRsnUser_InvVersion **Meaning**: The version for the parameter list specified is not valid. **Action**: Check for possible storage overlay.
8	*xxxx*0802	**Equate Symbol**: HisMT_kRsnUser_InconsistentIntvArea **Meaning**: The HISMT call for this interval does not match the previous HISMT call for the provided interval area. **Action**: If this is the first invocation for this interval area, follow the protocol described in "Programming requirements" on page 251 to provide an interval area. If this is a subsequent call to HISMT for the interval area, make sure all subsequent calls to HISMT with that interval area request the same MT metrics.

Table 27. Return and reason codes for the HISMT macro (continued)

Return code	Reason code	Equate symbol, meaning, and action
8	*xxxx*0803	**Equate Symbol**: HisMT_kRsnUser_IntvLenTooSmall **Meaning**: The interval area is less than HisMT_Hdr_kLength bytes. **Action**: Ensure the interval area length and the storage provided for the interval area is at least HisMT_Hdr_kLength bytes long. (See "Programming requirements" on page 251 for the protocol for obtaining big enough storage to contain the requested MT metrics)
8	*xxxx*0804	**Equate Symbol**: HisMT_kRsnUser_UnknownDataInIntvArea **Meaning**: The interval area for this HISMT call contains some unexpected data. A storage overlay may have occurred. **Action**: Issue HISMT request with a new interval area.
8	*xxxx*0805	**Equate Symbol**: HisMT_kRsnUser_IntvAreaNotAligned **Meaning**: The interval area provided for this HISMT call is not on a doubleword boundary. **Action**: Make sure that the interval area is on a doubleword boundary.
8	*xxxx*0806	**Equate Symbol**: HisMT_kRsnUser_IntvLenNot8ByteMultiple **Meaning**: The interval area length provided for this HISMT call is not a multiple of 8-bytes. **Action**: Provide an interval area whose length is an 8-byte multiple.
8	*xxxx*0807	**Equate Symbol**: HisMT_kRsnUser_UnknownEyeCatcher **Meaning**: The eye catcher in the interval area for this HISMT call is unexpected. A storage overlay may have occurred. **Action**: Issue a HISMT request with a new interval area. The first byte must contain binary zeroes.
10	–	**Equate Symbol**: HisMT_kRetUnknown **Meaning**: Unexpected failure. **Action**: Refer to the action provided with the specific reason code.
10	*xxxx*1001	**Equate Symbol**: HisMT_kRsnUnknown_Unknown **Meaning**: Unexpected failure. The state of the request is unpredictable. **Action**: Contact your system programmer.

Example

Operation

Requesting CAPCLASS, MAXCAPCLASS, PRODCLASS and PRODCORE metrics with HISMT:

1. Invoke HISMT with the requested metrics and pass an interval area that is the minimum interval area length, HisMT_Hdr_kLength.
2. Use the HisMT_Hdr_LengthRequired field in the interval area returned by the HISMT service and, allocate a new interval area that is at least HisMT_Hdr_LengthRequired bytes long.
3. Invoke HISMT for the first time with the requested metrics and pass the new interval area with sufficient storage.
4. Invoke HISMT for a subsequent call with the same requested metrics and pass the interval area from the previous invocation, unchanged. The requested HISMT metrics will be returned in the interval area for the current interval.

The code is as follows:

```
*********************************************************************
*   Invoke the HISMT service with the requested metrics and pass    *
*   an interval area that is the minimum interval area length,      *
*   HisMT_Hdr_kLength.                                              *
*   Note:                                                           *
*   On the first call, the first byte of the interval area          *
*   (HisMT_Hdr_EyeCatcherFirstChar) must contain binary zeroes      *
*********************************************************************
         LA    R8,LHisMT_Header
         Using HisMT_Hdr,R8
         MVI   HisMT_Hdr_EyeCatcherFirstChar,X'00'
         HISMT MF=(E,serviceList),INTVAREA=LHisMT_Header,              X
               INTVAREALEN=HisMT_Hdr_kLength,                          X
               PRODCLASS=YES,PRODCORE=YES,                             X
               MAXCAPCLASS=YES,CAPCLASS=YES,                           X
               RETCODE=LRetCode,RSNCODE=LRsnCode

*********************************************************************
*    Check return and reason code.                                  *
*   If HISMT requests a larger area, save required length.          *
*   Obtain new storage with required length (code not shown)        *
*********************************************************************
         LHI   R2,HisMT_kRetWarn
         L     R3,LRetCode
         CLR   R2,R3
         JNE   INTVAREA_UNEXP
         LHI   R4,HisMT_kRsnWarn_IntvAreaSmall
         L     R5,LRsnCode
         CLR   R4,R5
         JE    INTVAREA_WARN

INTVAREA_UNEXP   DS 0H
*
* Place code to handle unexpected return/reason codes here
*

INTVAREA_WARN  DS 0H
         L     R7,HisMT_Hdr_LengthRequired

*
* Place code to obtain storage of length HISMT_HDR_LENGTHREQUIRED and
* save address in R8
*

*********************************************************************
*   Invoke the HISMT service with the requested metrics and pass    *
*   an interval area that is the required interval area length,     *
*   HISMT_HDR_LENGTHREQUIRED. Save required interval area length    *
*   in LHisMT_INTVAREA_LEN                                          *
*   Note:                                                           *
*   On the first call, the first byte of the interval area          *
*   (HisMT_Hdr_EyeCatcherFirstChar) must contain binary zeroes      *
*********************************************************************
         ST    R7,LHisMT_INTVAREA_LEN
         Using HisMT_Hdr,R8
         MVI   HisMT_Hdr_EyeCatcherFirstChar,X'00'

EVN_LOOP      DS 0H
         HISMT MF=(E,serviceList),INTVAREA=HisMT_Hdr,                  X
               INTVAREALEN=LHisMT_INTVAREA_LEN,                        X
               PRODCLASS=YES,PRODCORE=YES,                             X
               MAXCAPCLASS=YES,CAPCLASS=YES,                           X
               RETCODE=LRetCode,RSNCODE=LRsnCode
```

```
*
* Place code to check return/reason codes here
*

*********************************************************************
*   On a subsequent HISMT call, invoke the HISMT service           *
*   with the same requested MT metrics passing the interval area   *
*   from the previous invocation, unchanged.                       *
*********************************************************************
*
* For a subsequent HISMT call, place code to process the returned metrics
* for the current interval here
*
         J    EVN_LOOP


DynArea     DSECT
LRetCode    DS    F
LRsnCode    DS    F
LHisMT_Header DS CL(HisMT_Hdr_kLength)
LHisMT_INTVAREA_LEN DS F
         HISMT MF=(L,serviceList)

         POP   USING
         HISYMT
```

Chapter 23. HISSERV macro — HISSERV Service

Description

HISSERV provides an interface to begin profiling and retrieve instrumentation data from the system. There are currently two types of instrumentation data:

Events
Events are recorded at the CPU or core level. As events occur, they are captured and recorded, to be queried at any interval determined by software. Events are grouped into event types, which can be enabled and disabled independently of each other.

Sampling
At predetermined intervals, a sample representing the current state of a CPU is stored into a Sampling Data Buffer (SDBs). As SDBs are filled software is notified allowing the software to process the full SDBs. The SDBs are then cleared to be reused by the hardware. There are different sampling types that can be enabled and disabled independently of each other, however a profiler can only indicate its intention to receive sampling data. The sampling frequency, as well as which sampling types are enabled are determined by the service parameters specified on a F hisproc,SERVICE command.

Specifically, with HISSERV you can do the following:

- Query for event info such as determining which events and event types are available. (REQUEST=QUERY,TYPE=EVENT).
- Query for sampling info such as the sampling interval and which sampling types are available. (REQUEST=QUERY,TYPE=SAMPLE).
- Query for statistics of whomever is currently profiling the system. (REQUEST=QUERY,TYPE=PROFILERS).
- Begin profiling the system, indicating to the system the intention of collecting one or more event types and/or sampling data. Requests that require a PROFILETKN must first use this to identify itself as wanting to profile the system. (REQUEST=PROFILE,ACTION=START)
- Stop profiling the system. The PROFILETKN is no longer useable and when the last profiler stops profiling the system, any unnecessary resources are released. (REQUEST=PROFILE,ACTION=STOP)
- Query for event data provided by the service. Requires a PROFILETKN. (REQUEST=QUERY,TYPE=EVENTDATA).

The HISSERV service is only enabled when the HIS address space has been initialized. The dynamic exit HIS.SERVSTAT can be used to be notified when the service has been enabled or disabled.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be holding any locks.

Environmental factor	Requirement
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided answer area (via the ANSAREA parameter) has the same requirements and restrictions as the control parameters. The user-provided CPU or core bitmask (via the CPUMASK parameter) has the same requirements and restrictions as the control parameters.

Programming Requirements

The caller should include the HISYSERV macro to get equate symbols for the return and reason codes.

The caller must include the HISYSERV macro to get a mapping of the output area provided via the ANSAREA parameter for REQUEST=QUERY.

The caller must include the HISYEXIT macro to get a mapping of the parameter area passed to the exit routine specified by the EXITRTN parameter for REQUEST=PROFILE,ACTION=START requests.

The caller must include the HISYSMPX macro to get a mapping of the parameter area passed to the exit routine specified by the EXITRTN parameter for REQUEST=PROFILE,ACTION=START requests, when the profiler requests sampling data (SAMPLE=YES).

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input Register Information

Before issuing the HISSERV macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the HISSERV macro, the caller does not have to place any information into any access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

The HISSERV macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede HISSERV.
HISSERV	
␣	One or more blanks must follow HISSERV.
REQUEST=PROFILE	
REQUEST=QUERY	
,ACTION=START	
,ACTION=STOP	
,OUTPROFILETKN=*outprofiletkn*	*outprofiletkn:* RS-type address or address in register (2) - (12)
,EVENT=*event*	*event:* RS-type address or address in register (2) - (12)
,EVENT=NO_EVENT	**Default:** EVENT=NO_EVENT
,SAMPLE=NO	**Default:** SAMPLE=NO
,SAMPLE=YES	

Syntax	Description
,NAME=*name*	*name:* RS-type address or address in register (2) - (12)
,EXITRTN=*exitrtn*	*exitrtn:* RS-type address or address in register (2) - (12)
,PROFILETKN=*profiletkn*	*profiletkn:* RS-type address or address in register (2) - (12)
,ANSAREA=*ansarea*	*ansarea:* RS-type address or address in register (2) - (12)
,ANSLEN=*anslen*	*anslen:* RS-type address or address in register (2) - (12)
,TYPE=EVENTDATA	
,TYPE=EVENT	
,TYPE=SAMPLE	
,TYPE=PROFILERS	
,PROFILETKN=*profiletkn*	*profiletkn:* RS-type address or address in register (2) - (12)
,CPUMASK=*cpumask*	*cpumask:* RS-type address or address in register (2) - (12)
,CPUMASK=ALL	**Default:** CPUMASK=ALL
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15),
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (0) or (2) - (12), (00), (GPR
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the HISSERV macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PROFILE
REQUEST=QUERY
A required parameter, used to indicate the type of request.

REQUEST=PROFILE
indicates the intention to start or stop profiling the system.

REQUEST=QUERY
indicates to query the service.

,ACTION=START
,ACTION=STOP
When REQUEST=PROFILE is specified, a required parameter, used to indicate the profiling action to take. Note that a PROFILE action cannot be requested from within a profiler's exit routine, nor should it be requested from a work unit holding a resource required by a profiler's exit routine.

,ACTION=START
indicates to start profiling the system.

,ACTION=STOP
indicates to stop profiling the system. When profiling for sampling data, the exit routine specified by EXITRTN will receive a final sampling related callback, with the HisSmpParmFlgs_Last flag on, for each CPU that is currently sampling. Note this service call will not return until after the EXITRTN has handled every CPU's final sampling callback. If no more profilers are profiling the system, all resources associated with profiling will be released.

,OUTPROFILETKN=*outprofiletkn*
When ACTION=START and REQUEST=PROFILE are specified, a required output parameter, into which the unique profiler token to identify this profiler will be returned. If this is the first profiler of the system, resources associated with profiling will be obtained and held until the last profiler stops profiling the system.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,EVENT=*event*
,EVENT=NO_EVENT
When ACTION=START and REQUEST=PROFILE are specified, an optional input parameter, which should contain the event types to profile. When querying the event data, only the event types specified when starting to PROFILE the system will be returned. If the event types are not authorized at the time of the PROFILE request the request will be remembered. Later if the system becomes authorized for that event type, it will be returned in any subsequent event data query. The storage is mapped by HisEvnTyp in macro HISYSERV and must be a subset of the event type data returned in HisEvn_ValidEvnTyp, which is returned in the REQUEST=QUERY,TYPE=EVENT request. If EVENT=NO_EVENT or the storage passed in is binary zeroes, the profiler will not be able to query for event data. The default is NO_EVENT.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32 bit field.

,SAMPLE=NO
,SAMPLE=YES
When ACTION=START and REQUEST=PROFILE are specified, an optional parameter, which is used to determine whether to receive callbacks with sampling data. When SAMPLE=YES, the exit routine defined in the EXITRTN parameter will be called as Sampling Data Blocks (SDBs) become available If sampling is not authorized at the time of the PROFILE request the request will be remembered. Later if the system becomes authorized for sampling, it will begin providing sampling data. The types of sampling entries returned by the service is dependent on the configuration of the service by the

SAMPTYPE parameter from the most recent F HIS,SERVICE or F HIS,BEGIN command. The default is SAMPLE=NO.

,SAMPLE=NO
Do not receive callbacks with sampling data.

,SAMPLE=YES
Receive callbacks with sampling data.

,NAME=*name*
When ACTION=START and REQUEST=PROFILE are specified, a required input parameter, which should contain the unique name identifying this profiler. The name should use EBCDIC characters from among the set of alphanumerics. The NAME will be returned in any QUERY,TYPE=PROFILE queries to identify this profiler. This name will also be displayed as output from the D HIS command. The NAME specified should be one that easily identifies the product requesting the profiling, for example the HIS supplied profiler starts with "HIS". The service will not allow another name starting with "HIS". The NAME must be unique for each profiler registered with the service.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,EXITRTN=*exitrtn*
When ACTION=START and REQUEST=PROFILE are specified, a required input parameter, which should contain the name of the exit routine that will be called when the service needs to notify the profiler for some reason, such as for sampling callbacks. The exit routine must reside in LPA, the LNKLST LNKLST concatenation, or the nucleus. The interface to the exit routine is described in macro HISYEXIT. The EXITRTN must be unique for each profiler registered with the service.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,PROFILETKN=*profiletkn*
When ACTION=STOP and REQUEST=PROFILE are specified, a required input parameter, which should contain the profiler's unique token received from this profiler's REQUEST=PROFILE,ACTION=START request (parameter OUTPROFILETKN.) The profiler's token will no longer be useable.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,ANSAREA=*ansarea*
When REQUEST=QUERY is specified, a required input parameter, which will be used by the service to store information associated with the query request. Macro HISYSERV contains mappings of the answer areas to provide, the size of the area depends on the type of query being requested. The minimum amount of storage required for the request to be successful and return a subset of the data, is HisAns_kLength.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSLEN=*anslen*
When REQUEST=QUERY is specified, a required input parameter, which should contain the length of the provided answer area. The length depends on the query that is requested. The minimum ANSLEN required for the request to be successful and return a subset of the data, is HisAns_kLength.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field, or specify a literal decimal value.

,TYPE=EVENTDATA
,TYPE=EVENT
,TYPE=SAMPLE
,TYPE=PROFILERS
When REQUEST=QUERY is specified, a required parameter, used to indicate the type of query to process.

,TYPE=EVENTDATA
indicates to process a query event data request. ANSAREA should point to storage that will be mapped by HisEvnData in macro HISYSERV. It is possible for the amount of storage required to change between two consecutive TYPE=EVENTDATA queries, depending on which event types are currently authorized in the system and which CPUs or cores are currently online.

,TYPE=EVENT
indicates to process a query event request. ANSAREA should point to storage that will be mapped by HisEvn in macro HISYSERV.

,TYPE=SAMPLE
indicates to process a query sample request. ANSAREA should point to storage that will be mapped by HisSmp in macro HISYSERV.

,TYPE=PROFILERS
indicates to process a query profiler info request. ANSAREA should point to storage that will be mapped by HisProf in macro HISYSERV. It is possible for the amount of storage required to change between two consecutive TYPE=PROFILERS queries, depending on the current number of profilers in the system.

,PROFILETKN=*profiletkn*
When TYPE=EVENTDATA and REQUEST=QUERY are specified, a required input parameter, which should contain the profiler's unique token received from this profilers's REQUEST=PROFILE,ACTION=START request (parameter OUTPROFILETKN).

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,CPUMASK=*cpumask*
,CPUMASK=ALL
When TYPE=EVENTDATA and REQUEST=QUERY are specified, an optional input parameter representing a bitmask of which CPUs and/or cores to query event data. The bitmask should be ECVTMaxMPNumBytesInMask bytes long. Bit 0 represents CPU 0's event data as well as core 0's event data to query, and so forth up to the bit position at CVTMAXMP. If requesting all CPUs and all cores specify ALL, pass in a CPUMASK of binary ones, or omit the CPUMASK parameter. The default is ALL.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND Codes

None.

Return and Reason Codes

When the HISSERV macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro HISYSERV provides equate symbols for the return and reason codes. Note carefully that bits 0-15 of the reason code may contain component-diagnostic data and must not be assumed to be 0.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 28. Return and Reason Codes for the HISSERV Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: Hisserv_kRetOk **Meaning**: HISSERV request successful. **Action**: Processing continues.
4	—	**Equate Symbol**: Hisserv_kRetWarn **Meaning**: Warning **Action**: Refer to the action provided with the specific reason code.
4	xxxx0401	**Equate Symbol**: Hisserv_kRsnWarn_AnsAreaSmall **Meaning**: For REQUEST=QUERY, the answer area provided was large enough to hold the minimum amount of data required for the request, but not large enough to hold all of the data requested. **Action**: Obtain a larger answer area using the HisAns_LengthRequire field returned in the request's ANSAREA.
8	—	**Equate Symbol**: Hisserv_kRetUser **Meaning**: HISSERV request failed due to a user error. **Action**: Refer to the action provided with the specific reason code
8	xxxx0801	**Equate Symbol**: Hisserv_kRsnUser_BadParmArea **Meaning**: Unable to access parameter area. **Action**: Check for possible storage overlay.
8	xxxx0802	**Equate Symbol**: Hisserv_kRsnUser_BadParmAreaALET **Meaning**: Bad parameter area ALET. **Action**: Make sure that the ALET associated with the parameter area is valid. The access register might not have been set up correctly.
8	xxxx0803	**Equate Symbol**: Hisserv_kRsnUser_BadVersion **Meaning**: Bad version for the parameter list was specified. **Action**: Check for possible storage overlay.
8	xxxx0804	**Equate Symbol**: Hisserv_kRsnUser_SrbMode **Meaning**: This function is only available in task mode. **Action**: Use function in task mode.
8	xxxx0805	**Equate Symbol**: Hisserv_kRsnUser_NotEnabled **Meaning**: This function is only available to enabled programs. **Action**: Use function while enabled.

Table 28. Return and Reason Codes for the HISSERV Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0806	**Equate Symbol**: Hisserv_kRsnUser_LocksHeld **Meaning**: This function is only available to unlocked programs. **Action**: Use function while unlocked.
8	xxxx0807	**Equate Symbol**: Hisserv_kRsnUser_CallerFRR **Meaning**: This function is only available to programs that have not established an FRR. **Action**: Retry the request without an FRR established.
8	xxxx0808	**Equate Symbol**: Hisserv_kRsnUser_BadRequest **Meaning**: A Bad request was made to the service. **Action**: Check for possible storage overlay.
8	xxxx0809	**Equate Symbol**: Hisserv_kRsnUser_BadProfTkn **Meaning**: Token specified was not a valid token. **Action**: Use a valid token provided by the REQUEST=PROFILE,ACTION=START request.
8	xxxx080A	**Equate Symbol**: Hisserv_kRsnUser_NameInUse **Meaning**: The name requested is already in use. **Action**: Provide a NAME that is unique to the service.
8	xxxx080B	**Equate Symbol**: Hisserv_kRsnUser_InvName **Meaning**: The name requested is invalid. **Action**: Provide a valid NAME, it cannot begin with HIS.
8	xxxx080C	**Equate Symbol**: Hisserv_kRsnUser_ExitRtnNotFound **Meaning**: The exit routine specified wasn't found. **Action**: Ensure the exit routine specified exists in LPA, the LNKLS concatenation, or the nucleus.
8	xxxx080D	**Equate Symbol**: Hisserv_kRsnUser_ExitRtnInUse **Meaning**: The exit routine specified is already in use. **Action**: A different exit routine must be provided.
8	xxxx080E	**Equate Symbol**: Hisserv_kRsnUser_BadEvnTyp **Meaning**: For REQUEST=PROFILE,ACTION=START requests, one or more event types specified could not be properly configured for because it is not allowed. Only event types returned in the HisEvn_ValidEvnTyp field of a REQUEST=QUERY,TYPE=EVENT request can be requested. **Action**: Ensure the event types being requested are a subset of the event types returned in the HisEvn_ValidEvnTyp field of a REQUEST=QUERY,TYPE=EVENT request.

Table 28. Return and Reason Codes for the HISSERV Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx080F	**Equate Symbol**: Hisserv_kRsnUser_BadProfReq **Meaning**: A bad PROFILE request was made to the service. **Action**: Check for possible storage overlay.
8	xxxx0810	**Equate Symbol**: Hisserv_kRsnUser_BadProfStart **Meaning**: For REQUEST=PROFILE,ACTION=START, a bad request was made. At least one event type or sampling should be requested when starting to profile the system. **Action**: Request at least one event type or sampling.
8	xxxx0811	**Equate Symbol**: Hisserv_kRsnUser_BadQuery **Meaning**: For REQUEST=QUERY, a bad query was requested. **Action**: Check for possible storage overlay.
8	xxxx0812	**Equate Symbol**: Hisserv_kRsnUser_BadAnsArea **Meaning**: For REQUEST=QUERY, unable to access answer area. **Action**: Provide a valid answer area.
8	xxxx0813	**Equate Symbol**: Hisserv_kRsnUser_BadAnsAreaALET **Meaning**: Bad answer area ALET. **Action**: Make sure that the ALET associated with the answer area is valid. The access register might not have been set up correctly.
8	xxxx0814	**Equate Symbol**: Hisserv_kRsnUser_AnsLenTooSmall **Meaning**: For REQUEST=QUERY, the answer area length is incorrect. **Action**: Ensure the answer area length and the storage provided as the answer area is at least HisAns_kLength bytes long.
8	xxxx0815	**Equate Symbol**: Hisserv_kRsnUser_BadCpuMask **Meaning**: For REQUEST=QUERY,TYPE=EVENTDATA requests, unable to access the CPU mask. **Action**: Provide a valid CPU mask.
8	xxxx0816	**Equate Symbol**: Hisserv_kRsnUser_BadCpuMaskALET **Meaning**: Bad CPU mask ALET. **Action**: Make sure that the ALET associated with the CPU mask is valid. The access register might not have been set up correctly.
8	xxxx0817	**Equate Symbol**: Hisserv_kRsnUser_NoEvnTyp **Meaning**: For REQUEST=QUERY,TYPE=EVENTDATA requests, the profiler making the request is not profiling events. **Action**: When registering with the system to profile, indicate the intention to profile events using the EVENT= parameter.

Table 28. Return and Reason Codes for the HISSERV Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
8	xxxx0818	**Equate Symbol**: Hisserv_kRsnUser_InvProfChange **Meaning**: A REQUEST=PROFILE request was made from a profiler's exit routine. **Action**: A REQUEST=PROFILE request cannot be made from a profiler's exit routine.
C	—	**Equate Symbol**: Hisserv_kRetEnv **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	xxxx0C01	**Equate Symbol**: Hisserv_kRsnEnv_NotAvailable **Meaning**: Function is not available. **Action**: This function is only available when the HIS address space is running.
C	xxxx0C02	**Equate Symbol**: Hisserv_kRsnEnv_NotReady **Meaning**: Function is available but is not currently ready to accept requests. **Action**: Retry the request.
C	xxxx0C03	**Equate Symbol**: Hisserv_kRsnEnv_NoStorage **Meaning**: There was not enough storage in HIS private storage to complete the request. **Action**: Contact your system programmer.
10	—	**Equate Symbol**: Hisserv_kRetUnk **Meaning**: Unexpected failure. **Action**: Refer to the action provided with the specific reason code.
10	xxxx1001	**Equate Symbol**: Hisserv_kRsnUnk_Unk **Meaning**: Unexpected failure. The state of the request is unpredictable. **Action**: Contact your system programmer.
10	xxxx1002	**Equate Symbol**: Hisserv_kRsnUnk_QueryCpu **Meaning**: For REQUEST=QUERY,TYPE=EVENTDATA, while attempting to query a CPU's event data an unknown error occurred. **Action**: Contact your system programmer.

Example

Operation

1. Profile for all available event types, and sampling.

2. Query the event types on 1 minute intervals for an hour, ensuring all event data is returned each query.
3. Process Sampling Data Blocks (SDBs) as they become available.
4. Stop profiling the system.

The code is as follows.

```
************************************************************
* Query Events to determine what can be enabled.           *
************************************************************
         HISSERV MF=(E,serviceList),REQUEST=QUERY,          *
               ANSAREA=EvnAnsArea,ANSLEN=EvnAnsLen,         *
               TYPE=EVENT,RETCODE=LRetCode,                 *
               RSNCODE=LRsnCode
         ICM  R5,B'1111',LRetCode
         JZ   EVNAREA_GOOD
         LA   R4,Hisserv_kRetWarn
         CLR  R5,R4
         JE   EVNAREA_WARN
EVNAREA_BAD   DS 0H
*
* Place code to check bad return/reason codes here
*
EVNAREA_WARN  DS 0H
         L    R5,LRsnCode
         NILH R5,0
         CHI  R5,Hisserv_kRsnWarn_AnsAreaSmall
         JNE  EVNAREA_BAD
*
* Place code to handle obtaining more storage for the
* ANSAREA, and repeat the request if necessary.
*
EVNAREA_GOOD   DS 0H
         LA   R8,EvnAnsArea
         Using HisEvn,R8
************************************************************
* Start profiling the system. Output from the previous     *
* query is used as input to this query, specifically we    *
* want to profile all valid event types.                   *
************************************************************
         HISSERV MF=(E,serviceList),REQUEST=PROFILE,        *
               ACTION=START,OUTPROFILETKN=ProfToken,        *
               NAME=ProfName,EXITRTN=ExitMod,SAMPLE=YES,    *
               EVENT=HisEvn_ValidEvnTyp,RETCODE=LRetCode,   *
               RSNCODE=LRsnCode
*
* Place code to check return/reason codes here
*
* Place code to obtain storage of length HisEvnData_kLength,
* save address in R8, length in EvnDataAnsLen
*
EVN_LOOP      DS 0H
************************************************************
* Query the current state of the events                    *
************************************************************
         Using HisEvnData,R8
         HISSERV MF=(E,serviceList),REQUEST=QUERY,          *
               TYPE=EVENTDATA,PROFILETKN=ProfToken,         *
               CPUMASK=ALL,ANSAREA=HisEvnData,              *
               ANSLEN=EvnDataAnsLen,RETCODE=LRetCode,       *
               RSNCODE=LRsnCode
         ICM  R5,B'1111',LRetCode
         JZ   EVNDATAAREA_GOOD
         LA   R4,Hisserv_kRetWarn
         CLR  R5,R4
         JE   EVNDATAAREA_WARN
EVNDATAAREA_BAD   DS 0H
*
* Place code to check bad return/reason codes here
*
EVNDATAAREA_WARN  DS 0H
         L    R5,LRsnCode
         NILH R5,0
         CHI  R5,Hisserv_kRsnWarn_AnsAreaSmall
         JNE  EVNAREA_BAD
*
* Place code to free storage of length EvnDataAnsLen, then
* obtain new storage of length HisEvnData_Length, save address
* in R8, length in EvnDataAnsLen
*
```

```
         J    EVN_LOOP
EVNDATAAREA_GOOD DS 0H
*
* Place code to process the returned events here.
*
         L    R5,EvnQueryVal
         BCT  R5,EVNDONE
         ST   R5,EvnQueryVal
         STIMER WAIT,BINTVL=EvnIntv
         J    EVN_LOOP
EVNDONE       DS 0H
*************************************************************
* Stop profiling the system                                 *
*************************************************************
         HISSERV MF=(E,serviceList),REQUEST=PROFILE,         *
               ACTION=STOP,PROFILETKN=ProfToken,             *
               RETCODE=LRetCode,RSNCODE=LRsnCode
*
* Place code to check return/reason codes here
*
EvnIntv     DC    F'6000'       One minute interval between  *
                                event queries
ProfName    DC    CL8'SAMPLE01' External name for profiler
ExitMod     DC    CL8'SAMPEXRT' EXITRTN Name
            HISYSERV            Return code information and  *
                                ANSAREA mappings.
DynArea     DSECT
LRetCode    DS    F
LRsnCode    DS    F
ProfToken   DC    CL16'0'
EvnQueryVal DC    F'60'         Query events 60 times
EvnAnsArea  DS    XL(HisEvn_Len+HisEvnCtr_Len)
EvnAnsLen   DS    AD(HisEvn_Len+HisEvnCtr_Len)
EvnDataAnsLen DS  D
         HISSERV MF=(L,serviceList)
*
*
* HISEXRTN CSECT, the EXITRTN located in LPA,LNKLIST or the
* nucleus.
*
HISEXRTN    CSECT
            Using HisExitParm,R1
            CLI   HisExitParm_Func,HisExitParmFunc_kStat
            JNE   CHECK_SMP
            PUSH  USING
            USING HisStatParm,R1
*
* Place code to process any service actions
*
            J     DONE
            POP   USING
CHECK_SMP   DS    0H
            CLI   HisExitParm_Func,HisExitParmFunc_kSmp
            JNE   DONE
            PUSH  USING
            USING HisSmpParm,R1
*
* Place code to process the full SDBs
*
            POP   USING
DONE        DS    0H
            HISYEXIT
            HISYSMPX
```

Chapter 24. HSPSERV — Read from and write to a Hiperspace

Description

HSPSERV transfers data between virtual storage areas in address spaces and hiperspaces. It reads data from a hiperspace to an address space and it writes data to a hiperspace from an address space.

A hiperspace can be either a **standard hiperspace**, of which there are two types, shared and nonshared, or an **ESO** (expanded storage only) **hiperspace**:

- The nonshared standard hiperspace and the shared standard hiperspace are backed with real storage and, if necessary, auxiliary storage. Through the buffer area in the address space, your program can view or **scroll** through the hiperspace. HSPSERV SWRITE and HSPSERV SREAD transfer data to and from a standard hiperspace. You create a standard hiperspace through the HSTYPE=SCROLL parameter on the DSPSERV macro. The description of HSPSERV macro for standard hiperspaces begins on "Read and write services for standard hiperspaces" on page 277.
- The ESO hiperspace is backed only with real storage. It is a high-speed buffer area or **cache** for data that your program needs. HSPSERV CWRITE and HSPSERV CREAD transfer data to and from an ESO hiperspace. You create an ESO hiperspace through the HSTYPE=CACHE parameter on the DSPSERV macro. The description of the HSPSERV macro for ESO hiperspaces begins on "Read and write services for ESO hiperspaces" on page 284.

The STOKEN parameter identifies the specific hiperspace to be read from or written to. The HSPALET parameter specifies an optional ALET for the hiperspace. The RANGLIST parameter identifies one or more of the storage ranges in the address space and the one or more storage ranges in the hiperspace. A storage range consists of contiguous 4K byte blocks starting on a 4K byte boundary.

HSPSERV is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the parameters that are valid only for supervisor state or PSW key 0 through 7 programs: CREAD, CWRITE, ADDRSP, and KEEP. For more information about hiperspaces and data spaces see *z/OS MVS Programming: Extended Addressability Guide*.

Read and write services for standard hiperspaces

Environment

The requirements for the caller who specifies SREAD and SWRITE are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN **Note:** PASN=HASN=SASN is required for a nonshared standard hiperspace for which an ALET is not used (that is, the HSPALET parameter is omitted).
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held

Environmental factor	Requirement
Control parameters:	Must be in the caller's primary address space. If the caller's PSW key is not zero, the PSW key must match the storage key associated with the control parameters.

Programming requirements

- If you code the HSPALET parameter on the HSPSERV macro, you must first code the SYSSTATE macro to indicate the ASC mode of your program.
- If you code the HSPALET parameter on the HSPSERV macro, you must provide a 144-byte save area in the caller's primary address space.
- The range list must be addressable in the caller's primary address space.

Restrictions

If you code HSPALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

Input register information

Before issuing the HSPSERV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

However, if the caller specifies the HSPALET parameter:

- General purpose register (GPR) 13 must contain the address of a 144-byte save area. The save area must be in the caller's primary address space.
- Access register (AR) 13 must contain 0, regardless of whether the caller is in primary or AR address space control (ASC) mode.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

The following figure describes the characteristics and restrictions for the use of standard hiperspaces, the hiperspaces that allow your program to **scroll** through large areas of data.

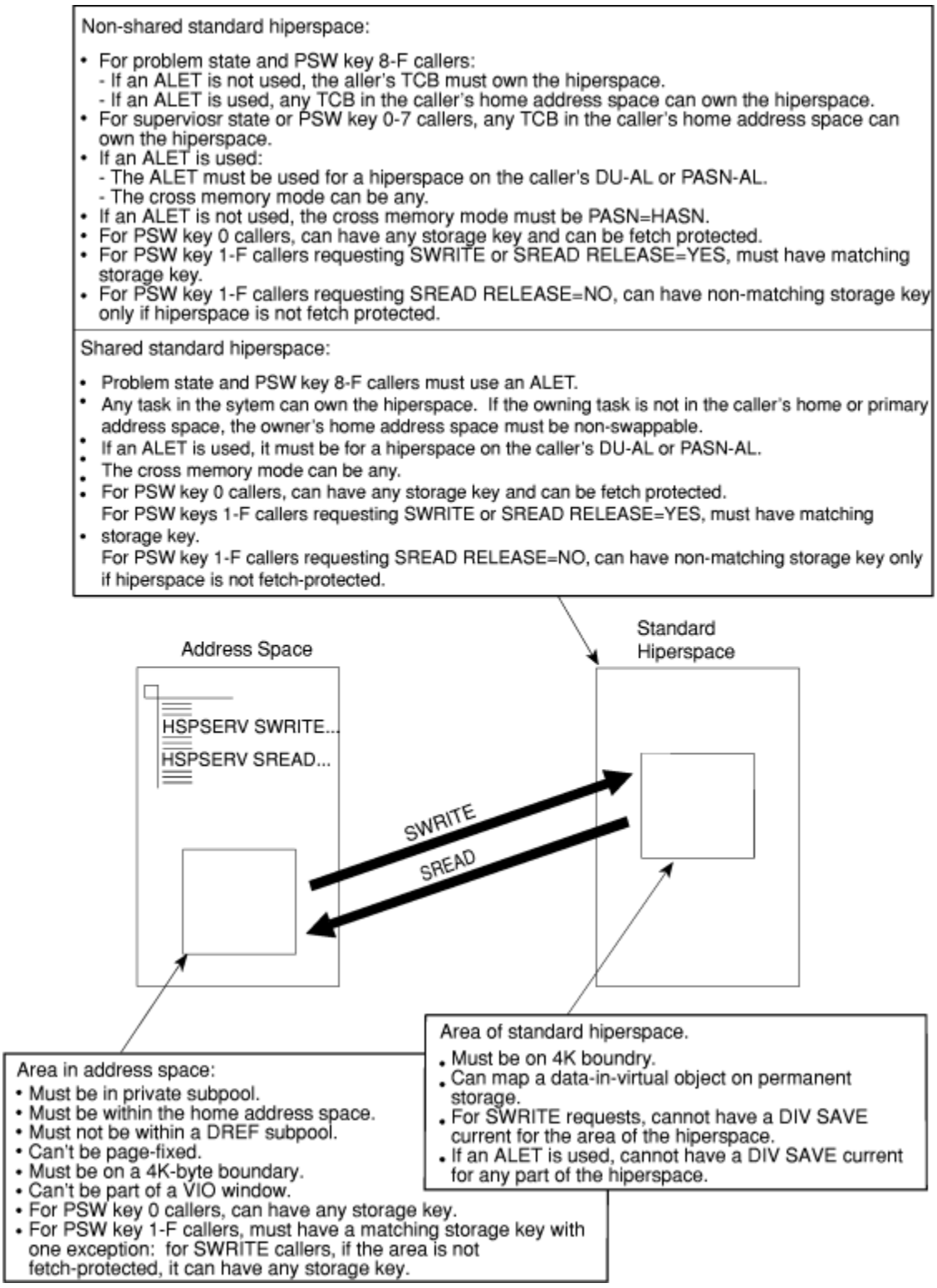


Figure 5. Characteristics and Restrictions for Standard Hiperspaces

Syntax

The standard form of the HSPSERV macro for standard hiperspaces is written as follows:

Syntax	Description

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	
ƀ	One or more blanks must follow HSPSERV.
SREAD	
SWRITE	
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=*n*	*n*: Number from 1 to 50.
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12). **Default**: NUMRANGE=1.
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - 12).
,RELEASE=NO	**Default**: RELEASE=NO.
,RELEASE=YES	
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

SREAD

Requests that the system read data from a standard hiperspace to an address space.

STOKEN and RANGLIST are required parameters on the SREAD request. HSPALET, NUMRANGE, RELEASE, RSNCODE, and RETCODE are optional parameters.

SWRITE
Requests that the system write data to a standard hiperspace from an address space.

Note:

1. When HSPSERV returns to the caller after the SWRITE operation, the contents of the address space storage range are not preserved. You can use the address space area again.
2. If the hiperspace maps a data-in-virtual object, do not issue an SWRITE request while a DIV SAVE request is current.

STOKEN and RANGLIST are required parameters on the SWRITE request. HSPALET, NUMRANGE, RETCODE, and RSNCODE are optional parameters.

,STOKEN=*stoken-addr*
Specifies the address of the eight-character variable that contains the STOKEN for the standard hiperspace from which the data is to be read or into which the data is to be written. Restrictions on standard hiperspaces are described in Figure 5 on page 280.

,HSPALET=*alet-addr*
Specifies either the address of a fullword or a register that contains the ALET for the hiperspace that is to be accessed. The ALET must be for a hiperspace that is on the caller's DU-AL or PASN-AL.

The HSPALET parameter is optional except for the following case: If the caller accesses a shared hiperspace, is in problem state and has PSW key 8 - F, HSPALET is required.

Use of the HSPALET parameter requires that the caller provide a 144-byte save area in the caller's primary address space. AR/GPR 13 must provide addressability to this area regardless of the caller's ASC mode. GPR 13 must contain the address of the area and AR 13 must contain 0.

If you code HSPALET, do not code RELEASE=YES.

If you code HSPALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

,NUMRANGE=*n*
,NUMRANGE=*num-addr*
Specifies the number of entries, from 1 to 50, or specifies a fullword that identifies the number of entries in the range list (that the RANGLIST parameter points to), or specifies a register containing the address of a fullword containing the number of entries. The default is NUMRANGE=1.

If you omit NUMRANGE, HSPSERV reads or writes one entry in the range list.

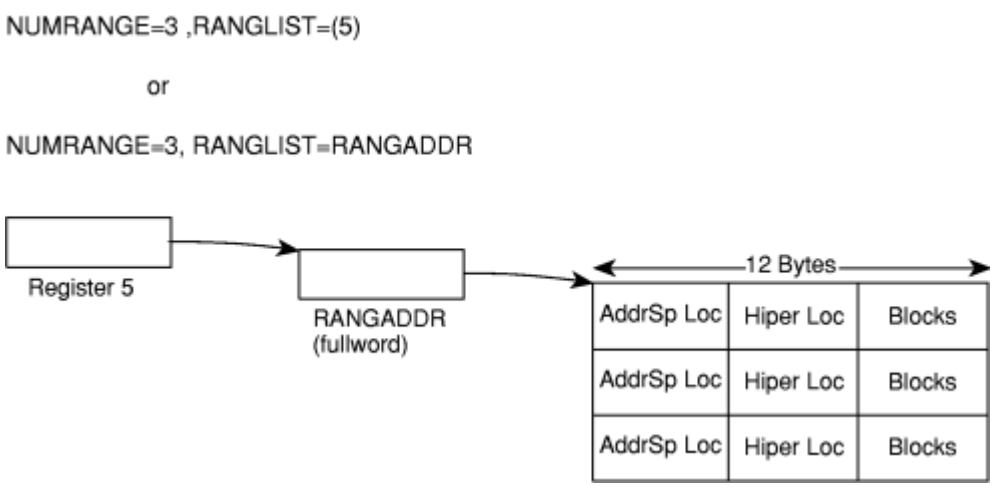
,RANGLIST=*list-addr*
Specifies a fullword that contains an **address of** a list of ranges (up to 50) that the system is to read or write, or specifies a register that contains the address of the fullword pointer to the range list. The range list consists of a number of entries (specified by NUMRANGE) where each entry consists of three words as follows:

First word
The starting virtual address in the address space into which the data is to be read or from which the data is to be written.

Second word
The starting virtual address in the hiperspace from which the system is to read or into which the system is to write.

Third word
The number of blocks the system is to read or write. Note that the address is the block number followed by 12 binary zeros.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:



Further restrictions on the areas in the address space and the hiperspace are described in Figure 5 on page 280.

On return, only if the caller issued the HSPSERV macro with the HSPALET parameter, the range list values might be different from the input values if the system could not at first successfully complete the read or write operation. In that case, the system changes the range list values, but does not restore the input values when it finally returns control to the caller.

,RELEASE=NO
,RELEASE=YES
Specifies whether or not the system is to release the hiperspace pages after it completes the SREAD operation. RELEASE is valid only with SREAD.

RELEASE=NO specifies that the system does not release the hiperspace pages after it completes the SREAD operation. Unless a subsequent SWRITE request changes the data, the same data will be available again on the next SREAD request. RELEASE=NO is the default.

RELEASE=YES specifies that, after the SREAD request, the system is to release the storage that backed the data in the hiperspace. If you code RELEASE=YES, do not code HSPALET.

,RETCODE=*ret-addr*
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsn-addr*
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the service.

ABEND codes

HSPSERV might abnormally terminate with abend code X'01D'. See *z/OS MVS System Codes* for an explanation of abend code X'01D'.

Return and reason codes

When control returns from HSPSERV SREAD or HSPSERV SWRITE, GPR 15 (and *ret-addr*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsn-addr*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Note: yy is X'09' for SREAD and X'0A' for SWRITE.

Table 29. Return and Reason Codes for HSPSERV SREAD and HSPSERV SWRITE

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: HSPSERV completed successfully. **Action**: None.
08	xxyy05xx	**Meaning**: System error. The system rejects the request. A hiperspace page is unavailable. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.
08	xxyy06xx	**Meaning**: System error. The system rejects the request. An address space page is unavailable. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.
0C	xx006xx	**Meaning**: System error. System failure because of environmental problems. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.

Read and write services for ESO hiperspaces

Environment

The requirements for the caller who requests CREAD and CWRITE are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW key 0 - 7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any
Control parameters:	The parameter list and range list must be in nonpageable, non-DREF storage. If the caller specifies HSPALET and is disabled, the save area must also be in nonpageable, non-DREF storage. The parameter list and save area must all be in the common area or in the private area of the caller's primary address space.

Programming requirements

- If you code the HSPALET parameter on the HSPSERV macro, you must first code the SYSSTATE macro to indicate the ASC mode of your program.
- If you code the HSPALET parameter on the HSPSERV macro, you must provide a 144-byte save area in the caller's primary address space.
- The range list must be addressable in the caller's primary address space.

Restrictions

If you code HSPALET, and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

Input register information

Before issuing the HSPSERV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

However, if the caller specifies the HSPALET parameter:

- General purpose register (GPR) 13 must contain the address of a 144-byte save area. The save area must be in the caller's primary address space.
- Access register (AR) 13 must contain 0, regardless of whether the caller is in primary or AR address space control (ASC) mode.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

The following figure describes the characteristics and restrictions for the use of ESO hiperspaces, the hiperspaces that act as a high-speed buffer or **cache** for data.

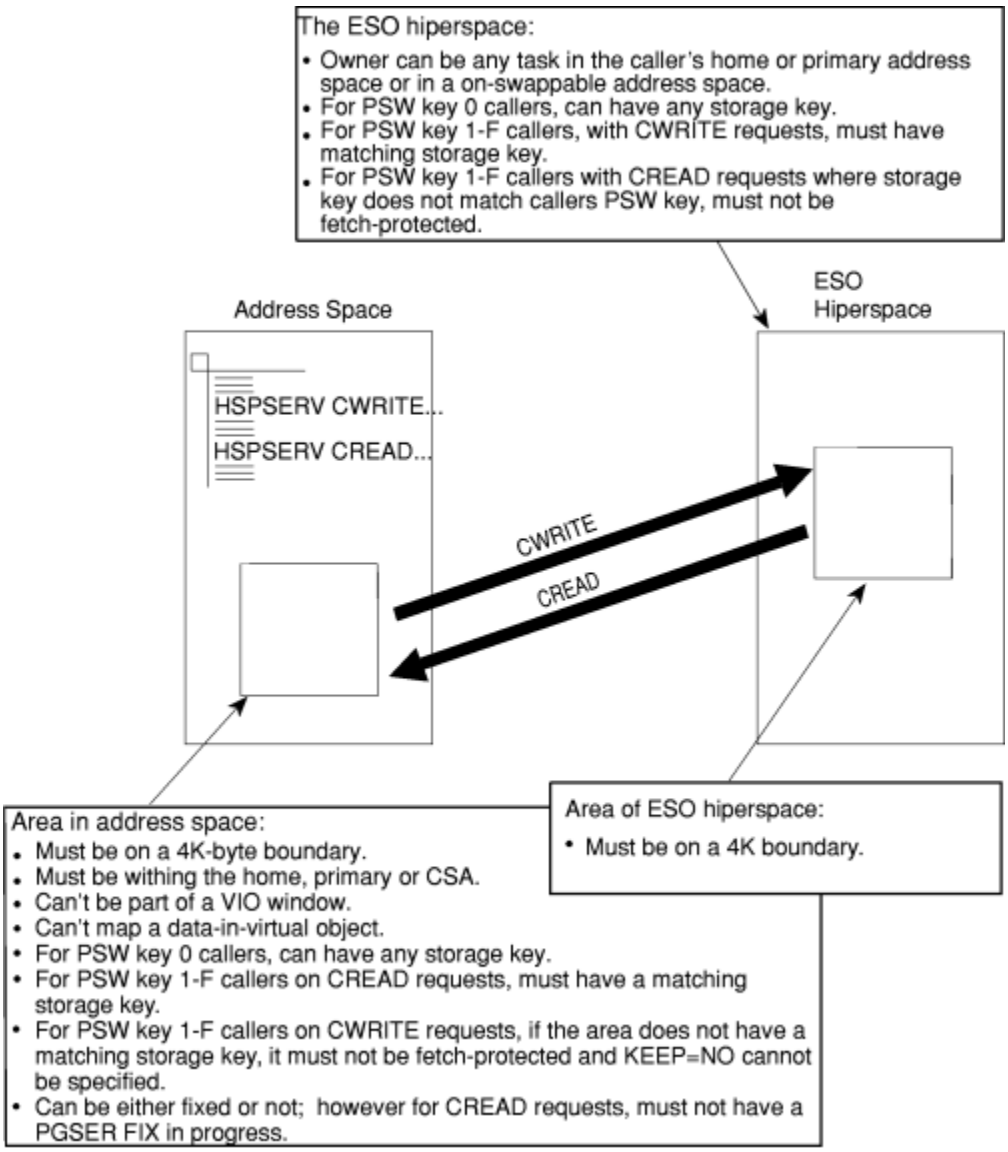


Figure 6. Characteristics and Restrictions for ESO Hiperspaces

Syntax

The standard form of the HSPSERV macro for ESO hiperspaces follows.

CAUTION: Code the parameters on the HSPSERV CREAD and HSPSERV CWRITE macros very carefully. Read the requirements for the address space buffer and the hiperspace, as listed in Figure 6 on page 286. For performance reasons, the system does not verify the location of the addresses you specify on these macros. Incorrect coding can cause damage to the system.

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	

Syntax	Description
ƀ	One or more blanks must follow HSPSERV.
CREAD	
CWRITE	
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=*n*	*n*: A number from 1 to 50.
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12). **Default**: NUMRANGE=1.
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).
,ADDRSP=HOME	**Default**: ADDRSP=HOME.
,ADDRSP=PRIMARY	
,ADDRSP=COMMON	
,KEEP=YES	**Default**: KEEP=YES.
,KEEP=NO	
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

CREAD

Requests that the system read data from an ESO hiperspace

If all blocks requested to be read are available in the hiperspace, then the system performs the read operation. However, if one or more of the blocks to be read are no longer available in the hiperspace, then the system returns a failing return code. (See return code 08.) In this case, the system does not tell you which blocks it successfully reads, if any.

STOKEN and RANGLIST are required parameters on the CREAD request. ADDRSP, NUMRANGE, RSNCODE, and RETCODE are optional parameters.

CWRITE
Requests that the system write data to an ESO hiperspace. If the system cannot write all the requested blocks to the hiperspace, then it doesn't write any and rejects the request. (See return code 08.) In this case, the data in the specified range in the hiperspace is unpredictable. Therefore, after an unsuccessful write, do not issue another CREAD against the failing hiperspace range of virtual storage until an intervening CWRITE is successful.

STOKEN and RANGLIST are required parameters on the CWRITE request. ADDRSP, NUMRANGE, KEEP, RSNCODE, and RETCODE are optional parameters.

,STOKEN=*stoken-addr*
Specifies the address of the 8-character variable that contains the STOKEN for the ESO hiperspace from which the data is to be read or into which the data is to be written. Restrictions on the hiperspace are described in Figure 6 on page 286.

,HSPALET=*alet-addr*
Specifies either the address of a fullword or a register that contains the ALET for the hiperspace that is to be accessed. The ALET must be for a hiperspace that is on the caller's DU-AL or PASN-AL.

Use of the HSPALET parameter requires that the caller provide a 144-byte save area in the caller's primary address space or in the common area. If the caller is disabled, the save area must be in nonpageable storage. AR/GPR 13 must provide addressability to this area regardless of the caller's ASC mode. GPR 13 must contain the address of the area and AR 13 must contain 0.

If you code HSPALET, do not code RELEASE=YES.

If you code HSPALET and you have an FRR recovery routine that gains control while HSPSERV is executing, your recovery routine cannot attempt retry at the time of error.

,NUMRANGE=*n*
,NUMRANGE=*num-addr*
Specifies a fullword that identifies the number of entries in the range list (that the RANGLIST parameter points to), or specifies a register containing the address of a fullword containing the number of entries, or specifies the number of entries, from 1 to 50. The default is NUMRANGE=1.

If you omit NUMRANGE, then HSPSERV reads or writes one virtual range.

,RANGLIST=*list-addr*
Specifies a fullword that contains the **address of** a parameter area in nonpageable storage that contains a list of up to 50 ranges that the system is to read or write, or specifies a register that contains the address of the fullword pointer to the range list.

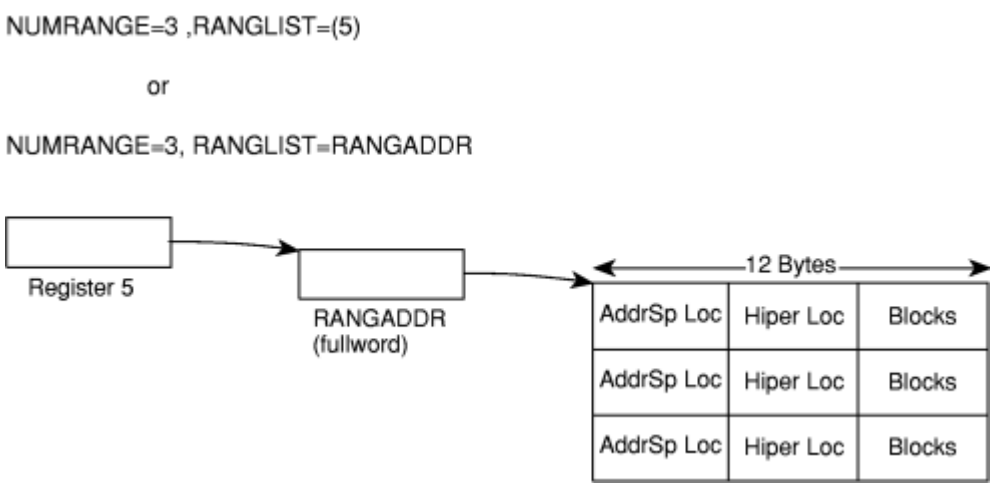
The range list consists of a number of entries (specified by NUMRANGE) where each entry consists of three words as follows:

First word
The starting virtual address in the address space into which the data is to be read or from which the data is to be written.

Second word
The starting virtual address in the hiperspace from which the system is to read or into which the system is to write.

Third word
The number of blocks the system is to read or write.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:



The one or more address space ranges on RANGLIST must be consistent with the ADDRSP parameter. When you specify ADDRSP=COMMON, each address space range described in the range list must reside entirely within CSA and have no intersections with other common area subpools or the private area. When you specify ADDRSP=HOME or ADDRSP=PRIMARY, each address space range described in the range list must reside entirely within the private area.

Restrictions on the areas in the address space and the hiperspace are described in Figure 6 on page 286.

The range list must be in the common area or in the private area of the caller's primary address space.

,ADDRSP=HOME
,ADDRSP=PRIMARY
,ADDRSP=COMMON

Specifies the location of the virtual storage range from which the system is to read or into which the system is to write. The location can be the caller's home address space (ADDRSP=HOME), the caller's primary address space (ADDRSP=PRIMARY), or the CSA (ADDRSP=COMMON). The default is ADDRSP=HOME.

,KEEP=YES
,KEEP=NO

Specifies whether or not the system preserves the source data in the virtual storage of the address space after it completes the CWRITE request. KEEP is valid only on the CWRITE request.

If you specify KEEP=YES, the data in the specified address space is unchanged and available for reference. The default is KEEP=YES.

If you specify KEEP=NO, the system might not preserve the data in the address space. If your program will reuse the same virtual storage area after the CWRITE request completes, use KEEP=NO.

,RETCODE=*ret-addr*

Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsn-addr*

Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

,MF=S

Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the macro service.

ABEND codes

HSPSERV might abnormally terminate with abend code X'01D'. See *z/OS MVS System Codes* for an explanation of abend code X'01D'.

Return and reason codes

When control returns from HSPSERV CREAD or HSPSERV CWRITE, GPR 15 (and *ret-addr*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsn-addr*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Note: yy is X'07' for CREAD and X'08' for CWRITE.

Table 30. Return and Reason Codes for HSPSERV CREAD and HSPSERV CWRITE

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: HSPSERV completed successfully. **Action**: None.
08	xxyy01xx	**Meaning**: Program error. The hiperspace data you requested is not available (CREAD request). **Action**: The data must be retrieved from its permanent copy.
08	xxyy02xx	**Meaning**: Program error. The system rejects the request because an address space page is not currently backed by real storage. You can repeat the HSPSERV request after you reference one or more pages, which causes the system to page the storage in CWRITE request. **Action**: Reference the page or pages that are not in processor storage.
08	xxyy03xx	**Meaning**: Environmental error. The system rejects the request because the necessary real storage frames are not currently available. **Action**: Rerun your program one or more times during a period of lower system usage. If the problem persists, consult your system programmer, who might be able to tune the system so that more resources are available to your program.
08	xxyy04xx	**Meaning**: Environmental error. The system rejects the request because no frames are currently available. **Action**: Rerun your program one or more times during a period of lower system usage. If the problem persists, consult your system programmer, who might be able to tune the system so that more resources are available to your program.
08	xxyy05xx	**Meaning**: System error. The system rejects the request because a hiperspace page is unavailable. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.
08	xxyy06xx	**Meaning**: System error. The system rejects the request because an address space page is unavailable. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.

HSPSERV - List form

Use the list form of the HSPSERV macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

The list form of the HSPSERV macro is written as follows:

Syntax	Description

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	
ƀ	One or more blanks must follow HSPSERV.
PLISTVER=*vernum*	*vernum*: Parameter list version 0 or 1
	Default: Version that allows all specified parameters.
,MF=(L,*list-addr)*	*list-addr*: Symbol.
,MF=(L,*list addr,attr)*	*attr*: 1- to 60-character input string. **Default**: 0D.

Parameters

Parameters for the list form of HSPSERV are as follows:

PLISTVER=*vernum*

Specifies the macro version associated with HSPSERV. PLISTVER is an optional parameter that determines which parameter list the system generates. Specify 0 if you use parameters only from this group:

- ADDRSP
- CREAD
- CWRITE
- KEEP
- MF
- NUMRANGE
- PLISTVER
- RANGLIST
- RELEASE
- RETCODE
- RSNCODE
- SREAD
- STOKEN
- SWRITE

If you use the HSPALET parameter, specify 1.

If you do not specify PLISTVER, the default is to allow all of the parameters you specify on the invocation to be processed.

,MF=(L,*list-addr*)
,MF=(L,*list-addr,attr*)
Specifies the list form of HSPSERV.

list-addr is the address of the storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

HSPSERV - Execute form

The execute form of the HSPSERV macro changes parameters in the control parameter list that the system created through the list form of the macro and performs the specified operation.

Syntax

The execute form of the HSPSERV macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	
ƀ	One or more blanks must follow HSPSERV.
SREAD	
SWRITE	
CREAD	
CWRITE	
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=1	**Default**: NUMRANGE=1.
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12).
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).

Syntax	Description
,RELEASE=NO	**Default**: RELEASE=NO.
,RELEASE=YES	
,ADDRSP=HOME	**Default**: ADDRSP=HOME.
,ADDRSP=PRIMARY	
,ADDRSP=COMMON	
,KEEP=YES	**Default**: KEEP=YES.
,KEEP=NO	
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).
,MF=(E,*list-addr,*COMPLETE)	*list-addr*: RX-type address or register (2) - (12).
,MF=(E,*list-addr*,NOCHECK)	**Default**: COMPLETE.

Parameters

The parameters are explained under the standard form of the HSPSERV macro with the following exceptions:

,MF=(E,*list-addr*,COMPLETE)
,MF=(E,*list-addr*,NOCHECK)
Specifies the execute form of the HSPSERV macro.

list-addr specifies the area that the system uses to store the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.

NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify.

HSPSERV - Modify form

Use the modify form of the HSPSERV macro together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

Syntax

The modify form of the HSPSERV macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	
ƀ	One or more blanks must follow HSPSERV.
SREAD	
SWRITE	
CREAD	
CWRITE	
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=1	**Default**: NUMRANGE=1.
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12).
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).
,RELEASE=NO	**Default**: RELEASE=NO.
,RELEASE=YES	
,ADDRSP=HOME	**Default**: ADDRSP=HOME.
,ADDRSP=PRIMARY	
,ADDRSP=COMMON	
,KEEP=YES	**Default**: KEEP=YES.
,KEEP=NO	
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).

Syntax	Description
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).
,MF=(E,*list-addr,*COMPLETE)	*list-addr*: RX-type address or register (2) - (12).
,MF=(E,*list-addr*,NOCHECK)	**Default**: COMPLETE.

Parameters

Parameters for the modify form of HSPSERV are described in the standard form of the macro with the following exceptions:

,MF=(M,*list-addr*,COMPLETE)
,MF=(M,*list-addr*,NOCHECK)
Specifies the modify form of the HSPSERV macro.

list-addr specifies the area that the system uses to store the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.

NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify.

Chapter 25. IARBRVEA — Verify virtual storage access (AR mode)

Description

Call the IARBRVEA service as a replacement for the TPROT instruction to determine whether a page of virtual storage can be accessed when the page to be tested resides in ALET-qualified storage.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, any PSW key (The key is used to determine storage access.)
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	AR
Interrupt status:	Enabled or disabled
Locks:	Any lock may be held.
Control parameters:	GPR 1 contains the virtual address of the page to be tested. AR 1 contains the ALET.

Programming requirements

- The IARBRVEA service is only available when the RCEOA46291APPLIED bit is set (B'1') in the RCE data area.
- The input virtual storage address in general purpose register (GPR) 1 and the ALET in access register (AR) 1 may refer to any address space.
- Include the IHAPVT macro. PVTBRVEA contains the address of the entry point of the routine.

Restrictions

None.

Input register information

Before calling the IARBRVEA service, the caller must ensure that the following GPRs contain the specified information:

Register
Contents

1
Virtual address to be tested

15
Address of the entry point

Before calling the IARBRVEA service, the caller must ensure that AR 1 contains the ALET.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system

1-14
Unchanged

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0
Used as a work register by the system

1-14
Unchanged

15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the IARBRVEA call as shown in the syntax diagram.

```
USING PSA,0
L     15,FLCCVT(0)
L     15,CVTPVTP-CVTMAP(15)
L     15,PVTEXTPT-PVT(15)
L     15,PVTBRVEA-PVTEXT(15)
BASR  14,15
```

Parameters

None.

ABEND codes

None.

Return and reason codes

When IARBRVEA returns control to your program, GPR 15 contains a return code. Table 31 on page 299 identifies return codes in hexadecimal, tells what each means, and recommends an action to take.

Table 31. Return codes for the IARBRVEA service

Hexadecimal return code	Meaning and action
00	**Meaning**: The caller has write access to the page and the page is not backed by a freemained frame. **Action:** None.
01	**Meaning**: The caller has read access to the page and the page is not backed by a freemained frame. **Action:** None.
02	**Meaning**: The caller has no access to the page and the page is not backed by a freemained frame. **Action:** None.
03	**Meaning**: The page either cannot be translated or is backed by a freemained frame. **Action:** Use VSMLOC, VSMLIST, or IARQDUMP to determine the status of the page.

Chapter 26. IARBRVER — Verify virtual storage access (primary address space)

Description

Call the IARBRVER service as a replacement for the TPROT instruction to determine whether a page of virtual storage can be accessed when the page to be tested resides in the primary address space.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, any PSW key (The key is used to determine storage access.)
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled
Locks:	Any lock may be held.
Control parameters:	GPR 1 contains the virtual address of the page to be tested.

Programming requirements

- The IARBRVER service is only available when the RCEOA46291APPLIED bit is set (B'1') in the RCE data area.
- The input virtual storage address refers to the primary address space at the time of invocation.
- Include the IHAPVT macro. PVTBRVER contains the address of the entry point of the routine.

Restrictions

None.

Input register information

Before calling the IARBRVER service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Virtual address to be tested

15
Address of the entry point

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Used as a work register by the system

1-14
Unchanged

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0
Used as a work register by the system

1-14
Unchanged

15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the IARBRVER call as shown in the syntax diagram.

```
USING PSA,0
L     15,FLCCVT(0)
L     15,CVTPVTP-CVTMAP(15)
L     15,PVTEXTPT-PVT(15)
L     15,PVTBRVER-PVTEXT(15)
BASR  14,15
```

Parameters

None.

ABEND codes

None.

Return and reason codes

When IARBRVER returns control to your program, GPR 15 contains a return code. Table 32 on page 302 identifies return codes in hexadecimal, tells what each means, and recommends an action to take.

Table 32. Return codes for the IARBRVER service

Hexadecimal return code	Meaning and action
00	**Meaning**: The caller has write access to the page and the page is not backed by a freemained frame. **Action:** None.

Table 32. Return codes for the IARBRVER service (continued)

Hexadecimal return code	Meaning and action
01	**Meaning**: The caller has read access to the page and the page is not backed by a freemained frame. **Action:** None.
02	**Meaning**: The caller has no access to the page and the page is not backed by a freemained frame. **Action:** None.
03	**Meaning**: The page either cannot be translated or is backed by a freemained frame. **Action:** Use VSMLOC, VSMLIST, or IARQDUMP to determine the status of the page.

Chapter 27. IARBRVKA — Verify virtual storage access

Description

Call the IARBRVKA service, as a replacement for the TPROT instruction to determine whether a page of virtual storage can be accessed under a specified key when the page to be tested resides in some ALET qualified storage.

Environment

The requirements for the caller are as follows.

Environmental factor	Requirement
Minimum authorization:	Supervisor state, any key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	AR
Interrupt status:	Enabled or disabled
Locks:	Any lock can be held
Control parameters:	R1 contains the virtual address of the page, AR1 contains the ALET. R2 contains the key that is used in bit positions 24-27

Programming requirements

The input VSA in R1 and ALET in AR1 refers to any address space. Include IHAPVT and IAXEPVT. Epvt_Brvka@ contains the address of the routine.

Restrictions

None.

Input register information

Register
Contents

1
Contains the virtual address to be tested, AR1 contains the ALET.

2
Contains the key to be used to test the storage in bit positions 24-27.

15
Contains the address of the entry point.

Output register information

Register
Contents

AR0, AR15, 0, 1
Can be used as work registers.

14
Contains the return address.

15
Contains the return code.

Performance implications

None.

Syntax

Write the IARBRVKA call as shown in the following syntax diagram.

```
L     15,FLCCVT-PSA(0)
L     15,CVTPVTP-CVTMAP(,15)
L     15,PVTEPV31-PVT(,15)
L     15,EPVT_BRVKA@-EPVT(,15)
BASR  14,15
```

Parameters

None.

ABEND codes

None.

Return and reason codes

Table 33. Return and Reason Codes for the IARBRVKA Callable service

Return Code	Meaning and Action
0	**Meaning:** The invoker has writer access to the page using the specified key and the page is not backed by a FREEMAINed frame. **Action:** None.
1	**Meaning:** The invoker has read access to the page using the specified key and the page is not backed by a FREEMAINed frame. **Action:** None.
2	**Meaning:** The invoker has no access to the page using the specified key and the page is not backed by a FREEMAINed frame. **Action:** None.
3	**Meaning:** Indeterminate state. Access to the page can or cannot result in an abend. **Action:** Use VSMLOC, VSMLIST, or IARQDUMP to determine the page's status.

Chapter 28. IARBRVKR — Replacement for the TPROT instruction

Description

Call the IARBRVKR service, as a replacement for the TPROT instruction, to determine whether a page of virtual storage can be accessed when the page to be tested is stored in the primary address space under a specified key.

Environment

The requirements for the caller are as follows.

Environmental factor	Requirement
Minimum authorization:	Supervisor state, any key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled
Locks:	Any lock can be held
Control parameters:	R1 contains the virtual address of the page to be tested. R2 contains the key used in bit positions 24-27

Programming requirements

The input VSA refers to the primary address space at the time of invocation. Include IHAPVT and IAXEPVT. Epvt_Brvkr@ contains the address of the routine.

Restrictions

None.

Input register information

Register
Contents

1
Contains the virtual address to be tested.

2
Contains the key to be used to test the storage in bit positions 24-27.

15
Contains the address of the entry point.

Output register information

Register
Contents

AR0, AR15, 0, 1
Can be used as work registers.

14
Contains the return address.

15
Contains the return code.

Performance implications

None.

Syntax

Write the IARBRVKR call as shown in the following syntax diagram.

```
L     15,FLCCVT-PSA(0)
L     15,CVTPVTP-CVTMAP(,15)
L     15,PVTEPV31-PVT(,15)
L     15,EPVT_BRVKR@-EPVT(,15)
BASR  14,15
```

Parameters

None.

ABEND codes

None.

Return and reason codes

Table 34. Return and Reason Codes for the IARBRVKR Callable service

Return Code	Meaning and Action
0	**Meaning:** The invoker has writer access to the page using the specified key and the page is not backed by a FREEMAINed frame. **Action:** None.
1	**Meaning:** The invoker has read access to the page using the specified key and the page is not backed by a FREEMAINed frame. **Action:** None.
2	**Meaning:** The invoker has no access to the page using the specified key and the page is not backed by a FREEMAINed frame. **Action:** None.
3	**Meaning:** Indeterminate state. Access to the page can or cannot result in an abend. **Action:** Use VSMLOC, VSMLIST, or IARQDUMP to determine the page's status.

Chapter 29. IARCP64 — 64-bit cell pool services

Description

Use IARCP64 to request 64-bit cell pool services.

With IARCP64, you can request to:

- Build a pool (REQUEST=BUILD).
- Obtain a cell from the pool (REQUEST=GET).
- Return a cell to the pool (REQUEST=FREE).
- Delete the pool (REQUEST=DELETE).

Note: There is diagnostic support for 64-bit cell pools in IPCS by way of the CBFORMAT command. CBF *cpid* STR(IAXCPHD) formats the cell pool header, where *cpid* is the cell pool identifier that was returned on IARCP64 REQUEST=BUILD. If you cannot locate your cell pool identifier in the dump, simply browse storage starting at X'100000000' and issue a FIND on CPHD. There are multiple cell pools, so you must look at the cell contents to make sure that you have the right pool. To see details about all of the cells in the pool, use the EXIT option as follows: CBF *cpid* STR(IAXCPHD) EXIT.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For IARCP64 REQUEST=BUILD, use of the COMMON=YES, TYPE=DREF, TYPE=FIXED, OWNINGTASK=RCT, MEMLIMIT=NO, or MOTKN parameter or the Key00ToF0 parameter with a value other than X'90', require any of the following: • Supervisor state • PSW key 0-7 • APF-authorized All other options have a minimum authorization of Problem state and PSW key 8-15. For IARCP64 REQUEST=GET, FREE or DELETE, the caller must be able to modify the storage for the cell pool. That means the caller must be in key 0 or in the same key as the cell pool or the cell pool must be in the public key (key 9). Supervisor state is required for the TRACE=YES option. APF authorization has no bearing on these services.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary or access register (AR)

Environmental factor	Requirement
Interrupt status:	For the BUILD and DELETE requests, enabled. For the GET and FREE requests: • The caller might be enabled or disabled for interrupts when requesting cells that are from pools are defined as COMMON=YES and TYPE=FIXED. • For all other combinations of options, the caller must be enabled for interrupts.
Locks:	For the BUILD and DELETE requests, no locks can be held. For the GET request, the following locks must be held by the caller or must be obtainable by IARCP64: • For requests with EXPAND=NO, the caller might hold locks but is not required to hold any. • For requests with COMMON=NO and EXPAND=YES, the caller might hold the local lock (LOCAL or CML) of the current primary address space. • For requests with COMMON=YES and EXPAND=YES, the locking restrictions for the caller are the same as for IARV64 REQUEST=GETCOMMON. For the FREE request, the caller might hold locks but is not required to hold any.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

Specify SYSSTATE AMODE64=YES before invoking this macro.

Restrictions

None.

Input register information

Before issuing the IARCP64 macro, the caller does not have to place any information into any general-purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or by using it as a base register.

Output register information

When control returns to the caller, the 64-bit GPRs contain:

For REQUEST=BUILD:

Register
Contents

0
Reason code in the low 32 bits if the return code is not 0. Otherwise, used as a work register by the system.

1
Used as a work register by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Return code in the low 32 bits.

For REQUEST=GET:

Register
Contents

0
Reason code in the low 32 bits if the return code is not 0. Otherwise, used as a work register by the system.

1
The address of the obtained cell.

2-12
Unchanged if REGS=SAVE was specified, used as work registers by the system if REGS=USE was specified.

13
Unchanged.

14
Used as a work register by the system.

15
Return code in the low 32 bits.

For REQUEST=FREE:

Register
Contents

0-1
Used as a work register by the system.

2-12
Unchanged if REGS=SAVE was specified, used as work registers by the system if REGS=USE was specified.

13
Unchanged.

14-15
Used as a work register by the system.

For REQUEST=DELETE:

Register
Contents

0-1
Used as a work register by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IARCP64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARCP64.
IARCP64	
␣	One or more blanks must follow IARCP64.
REQUEST=BUILD	
REQUEST=GET	
REQUEST=FREE	
REQUEST=DELETE	
,HEADER=*header*	*header:* RS-type address or address in register (2) - (12)
,CELLSIZE=*cellsize*	*cellsize:* RS-type address or address in register (2) - (12)
,OUTPUT_CPID=*output_cpid*	*output_cpid:* RS-type address or address in register (2) - (12)
,COMMON=NO	
,COMMON=YES	
,OWNINGTASK=CURRENT	
,OWNINGTASK=MOTHER	
,OWNINGTASK=IPT	
,OWNINGTASK=JOBSTEP	

Syntax	Description
,OWNINGTASK=CMRO	
,OWNINGTASK=RCT	
,MEMLIMIT=YES	**Default:** MEMLIMIT=YES
,MEMLIMIT=NO	
,MOTKN=*motkn*	*motkn:* RS-type address or address in register (2) - (12)
,MOTKN=NO_MOTKN	**Default:** MOTKN=NO_MOTKN
,DUMP=LIKERGN	
,DUMP=LIKELSQA	
,DUMP=LIKECSA	
,DUMP=LIKESQA	
,DUMP=NO	
,DUMPPRIO=*dumpprio*	*dumpprio:* RS-type address or address in register (2) - (12)
,OWNER=HOME	
,OWNER=PRIMARY	
,OWNER=SECONDARY	
,OWNER=SYSTEM	
,OWNER=BYASID	
,OWNINGASID=*owningasid*	*owningasid:* RS-type address or address in register (2) - (12)
,FPROT=YES	
,FPROT=NO	
,SENSITIVE=UNKNOWN	**Default:** SENSITIVE=UNKNOWN
,SENSITIVE=YES	
,SENSITIVE=NO	
,TYPE=PAGEABLE	
,TYPE=DREF	
,TYPE=FIXED	

Syntax	Description
,CALLERKEY=YES	
,CALLERKEY=NO	
,KEY00TOF0=*key00tof0*	*key00tof0:* RS-type address or address in register (2) - (12)
,TRAILER=COND	
,TRAILER=YES	
,TRAILER=NO	
,FAILMODE=RC	
,FAILMODE=ABEND	
,LOCALSYSAREA=NO	**Default:** LOCALSYSAREA=NO
,LOCALSYSAREA=YES	
,INPUT_CPID=*input_cpid*	*input_cpid:* RS-type address or address in register (2) - (12)
,CELLADDR=*celladdr*	*celladdr:* RS-type address or address in register (2) - (12)
,EXPAND=YES	
,EXPAND=NO	
,TRACE=YES	
,TRACE=NO	
,CELLNAME=*cellname*	*cellname:* RS-type address or address in register (2) - (12)
,CELLADDR=*celladdr*	*celladdr:* RS-type address or address in register (2) - (12)
,REGS=SAVE	
,REGS=USE	
,INPUT_CPID=*input_cpid*	*input_cpid:* RS-type address or address in register (2) - (12)
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12), (GPR15), (REG15), or (R15).

Syntax	Description
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12), (GPR0), (GPR00), (REG0), (REG00), or (R0).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1 that is the name on the IARCP64 macro invocation. The name must conform to the rules for an ordinary assembly language symbol.

REQUEST=BUILD
REQUEST=GET
REQUEST=FREE
REQUEST=DELETE
A required parameter that indicates the type of request.

REQUEST=BUILD
Request to build the pool. The initial pool size is 1 MB. The CELLSIZE and TRAILER specifications determine how many available cells are in the pool.

REQUEST=GET
Request to obtain a cell from the pool.

REQUEST=FREE
Request to return a cell to the pool.

Note: This request is unconditional and will abnormally end when a problem. No return and reason codes are provided; therefore, do not specify the RETCODE and RSNCODE parameters.

REQUEST=DELETE
Request to delete the pool.

Note: This request is unconditional and will abnormally end when a problem. No return and reason codes are provided; therefore, do not specify the RETCODE and RSNCODE parameters.

Parameters for REQUEST=BUILD

The following parameters are valid when you specify REQUEST=BUILD:

,HEADER=*header*
A required input parameter that specifies information to be placed into the pool header for potential diagnostic purposes. The information helps to identify the requester and the purpose for the pool.

To code: Specify the RS-type address, or address in register (2) - (12), of a 24-character field.

,CELLSIZE=*cellsize*
A required input parameter that indicates the size of a cell in the pool. The cell size can be anywhere between 1 and (1M-8192)/2 or 520,192 bytes. Cell size is rounded up to a quadword multiple for cell sizes less than a cache line. Cells larger than a cache line are rounded up to a cache line multiple. Cells larger than a page are rounded to start on a page boundary. The first cell in an extent is always on a page boundary. Specifying a cell size that is at least 4 bytes less than the size after rounding for boundary alignment makes room for a trailer to be inserted. See TRAILER=YES below.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,OUTPUT_CPID=*output_cpid*
A required output parameter that is to contain the cell pool ID.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,COMMON=NO
,COMMON=YES
A required parameter that indicates if the pool is to reside in common storage.

,COMMON=NO
The pool is not to reside in common storage.

,COMMON=YES
The pool is to reside in common storage.

,OWNINGTASK=CURRENT
,OWNINGTASK=MOTHER
,OWNINGTASK=IPT
,OWNINGTASK=JOBSTEP
,OWNINGTASK=CMRO
,OWNINGTASK=RCT
A required parameter that indicates the task to be considered as the owner of the cell pool. When this task ends, the cell pool is automatically deleted.

,OWNINGTASK=CURRENT
The current task is to be the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=MOTHER
The mother task of the current task is to be the owner. If the current task is the cross-memory resource owning task, the request fails. Do not specify this unless the program is in task mode.

,OWNINGTASK=IPT
The initial pthread task is to be the owner. If the current task or mother task is not the IPT, then this defaults to the current task as the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=JOBSTEP
The jobstep task of the current task (the task with TCB address in field TCBJSTCB of the current task's TCB) is to be the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=CMRO
The cross-memory resource-owning task of the current primary address space is to be the owner.

,OWNINGTASK=RCT
The region control task (RCT) of the current primary address space is to be the owner.

,MEMLIMIT=YES
,MEMLIMIT=NO
An optional parameter that specifies whether the 64-bit private memory objects created for this cell pool are to count towards the address space MEMLIMIT. The default is MEMLIMIT=YES.

,MEMLIMIT=YES
The 64-bit private memory objects contribute towards the address space MEMLIMIT.

,MEMLIMIT=NO
The 64-bit private memory objects are not counted against the address space MEMLIMIT.

,MOTKN=*motkn*
,MOTKN=NO_MOTKN
An optional input parameter that identifies the memory object token to be associated with the memory object. This is expected to be a memory object token that is user-generated (as opposed to having been created by the system with the OUTMOTKN parameter of IARV64 GETSTOR). The main reason to specify your own MOTKN is to have the cell pool extents be associated with other memory objects from a dumping perspective. WARNING: If you use this MOTKN on other IARV64 REQUEST=GETSTOR calls, a call to IARCP64 REQUEST=DELETE detaches all memory objects allocated with this MOTKN. Similarly, a call to IARV64 REQUEST=DETACH with this MOTKN results in detaching all extents of the cell pool, without deleting control information for the cell pool. Unpredictable behavior can result. The default is NO_MOTKN, which indicates that no memory object token is supplied to associate this memory object with others.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,DUMP=LIKERGN
,DUMP=LIKELSQA
,DUMP=LIKECSA
,DUMP=LIKESQA
,DUMP=NO
A required parameter that indicates how to dump this pool.

When COMMON=NO is specified:

,DUMP=LIKERGN
Dump the pool according to the rules for RGN.

,DUMP=LIKELSQA
Dump the pool according to the rules for LSQA.

,DUMP=NO
Do not dump the pool based on the RGN and LSQA SDATA options.

When COMMON=YES is specified:

,DUMP=LIKECSA
Dump the pool according to the rules for CSA.

,DUMP=LIKESQA
Dump the pool according to the rules for SQA.

,DUMP=NO
Do not dump the pool based on the CSA and SQA SDATA options.

,DUMPPRIO=*dumpprio*
When DUMP=LIKERGN, COMMON=NO and REQUEST=BUILD are specified, a required input parameter that contains the dump priority to be used when dumping the pool. The value can be in the range 1-99 with 1 being the highest priority. See the documentation for the GETSTOR option of the IARV64 macro for a discussion on dump priorities.

To code: Specify the RS-type address, or address in register (2) - (12), of a 1-byte field.

,OWNER=HOME
,OWNER=PRIMARY
,OWNER=SECONDARY
,OWNER=SYSTEM
,OWNER=BYASID
When COMMON=YES is specified, a required parameter that designates the owner of the storage.

,OWNER=HOME
The home address space is to be the owner.

,OWNER=PRIMARY
The primary address space is to be the owner.

,OWNER=SECONDARY
The secondary address space is to be the owner.

,OWNER=SYSTEM
The system is to be the owner. Use this only when there is no specific address space, which can be considered the owner.

,OWNER=BYASID
The owner is the ASID specified by the OwningASID parameter.

,OWNINGASID=*owningasid*
When OWNER=BYASID, COMMON=YES and REQUEST=BUILD are specified, a required input parameter that specifies the ASID that is to be the owner. A value of 0 is equivalent to having specified OWNER=SYSTEM.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field.

,FPROT=YES
,FPROT=NO
A required parameter that indicates whether the pool storage is to be fetch-protected.

,FPROT=YES
The pool storage is to be fetch-protected.

,FPROT=NO
The pool storage is not to be fetch-protected.

,SENSITIVE=UNKNOWN
,SENSITIVE=YES
,SENSITIVE=NO
An optional keyword input that specifies whether the pool will contain data that is potentially sensitive (for instance, personal or confidential). The default is SENSITIVE=UNKNOWN.

,SENSITIVE=UNKNOWN
Specifies that the sensitivity of the data is not known. The pool may or may not contain sensitive data.

,SENSITIVE=YES
Specifies that the pool contains sensitive data.

,SENSITIVE=NO
Specifies that the pool does not contain sensitive data.

,TYPE=PAGEABLE
,TYPE=DREF
,TYPE=FIXED
A required parameter that indicates the type of storage for the pool.

,TYPE=PAGEABLE
The pool storage is to be pageable.

,TYPE=DREF
The pool storage is to be disabled-reference (DREF).

,TYPE=FIXED
The pool storage is to be page-fixed.

,CALLERKEY=YES
,CALLERKEY=NO
A required parameter that indicates whether the pool storage is to be in the key of the caller of the BUILD request.

,CALLERKEY=YES
The pool storage is to be in the key of the caller.

,CALLERKEY=NO
The pool storage is not to be in the key of the caller, but instead in the key specified by the Key00ToF0 parameter.

,KEY00TOF0=*key00tof0*
When CALLERKEY=NO is specified, a required input parameter that indicates the key for the pool storage. The value should be in the range X'00' to X'F0' (for example, the key 0-15 in the high 4 bits of the byte) for a caller that is key 0-7, supervisor state, or APF-authorized. The value X'90' is the only accepted key for a caller that is key 8-15, problem state, and not APF-authorized.

To code: Specify the RS-type address, or address in register (2) - (12), of a 1-byte field.

,TRAILER=COND
,TRAILER=YES
,TRAILER=NO
A required parameter that indicates whether the cell is to have a trailer area after the user portion of the cell, which is set on GET processing and checked on FREE processing.

Note: Requesting a trailer can cause the cell size to be increased to provide room for the trailer. This increase in size occurs before rounding for boundary alignment. For example, requesting a cell size of 4096 and TRAILER=YES results in cells being 8192 bytes. If you do not need the entire 4096 bytes, specify a cell size of 4092 bytes and now the trailer fits in the same page.

,TRAILER=COND
The cell storage should have trailer processing in the following cases:

- When the service-rounded cell size has room for the trailer without requiring a larger cell to be allocated.
- When system diagnostic controls requests that trailers be appended to cells obtained by IARCP64. If this results in trailer processing, it works as described for TRAILER=YES below.

Note: The system diagnostic control for trailers in IARCP64 cell pools is examined at BUILD time only.

,TRAILER=YES
The cell storage is to have trailer processing. If the application writes past the end of the specified cell size, it will overrun the trailer. On a FREE request, this is detected and causes an ABEND.

,TRAILER=NO
The cell storage is not to have trailer processing, even if requested by way of a system diagnostic control.

,FAILMODE=RC
,FAILMODE=ABEND
A required parameter that indicates what to do if the request is not successful.

,FAILMODE=RC
The request should return with a failure return code when there are insufficient memory resources to satisfy the request. All errors in parameter specification or parameter access results in the request abnormally ending.

,FAILMODE=ABEND
The request should abnormally end when there are insufficient memory resources to satisfy the request.

,LOCALSYSAREA=NO
,LOCALSYSAREA=YES
When COMMON=NO is specified, an optional parameter that specifies whether this is an explicit allocation request for 64-bit virtual storage in the local system area. This parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is LOCALSYSAREA=NO.

,LOCALSYSAREA=NO
The request will not be satisfied from the local system area.

,LOCALSYSAREA=YES
The request is to be satisfied from the local system area. The storage obtained with this keyword will not be copied during fork processing. The use of local system area storage does not preclude checkpoint from succeeding.

Parameters for REQUEST=GET

The following parameters are valid when you specify REQUEST=GET:

,INPUT_CPID=*input_cpid*
A required input parameter that contains the cell pool ID returned on the successful BUILD request.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,CELLADDR=*celladdr*
An optional output parameter of the obtained cell. If CELLADDR is not specified, the cell address is left in register 1.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-byte pointer field.

,EXPAND=YES
,EXPAND=NO
A required parameter that indicates whether to attempt expanding the pool if there is no available cell.

,EXPAND=YES
Indicates that an attempt to expand the pool should be made. Each successful expansion results in a 1 MB increase in the pool size.

,EXPAND=NO
Indicates that no attempt to expand the pool should be made.

,TRACE=YES
,TRACE=NO
A required parameter that indicates whether the invocation is to be traced.

Note: Tracing is available only to supervisor state callers.

,TRACE=YES
The entry is to be traced. If you are running in supervisor state, use this option, unless performance needs dictate otherwise.

Note: TRACE=YES on GET also results in TRACE=YES on FREE, so if you use TRACE=YES, ensure that the FREE request is in supervisor state.

,TRACE=NO
The entry is not to be traced. You must use this option if running in problem state.

,FAILMODE=RC
,FAILMODE=ABEND
A required parameter that indicates what to do if the request is not successful.

,FAILMODE=RC
The request should return with a failure return code when there are insufficient memory resources to satisfy the request. All errors in parameter specification or parameter access results in the request abnormally ending.

,FAILMODE=ABEND
The request should abnormally end when there are insufficient memory resources to satisfy the request.

,REGS=SAVE
,REGS=USE
A required parameter that indicates how to deal with the registers.

,REGS=SAVE
The request should save and preserve the contents of 64-bit GPRs 2 - 12, starting at offset 40 in a 144-byte area pointed to by register 13.

,REGS=USE
The request can use registers 2 - 12.

Parameters for REQUEST=FREE

The following parameters are valid when you specify REQUEST=FREE:

,CELLNAME=*cellname*
,CELLADDR=*celladdr*
A required input parameter that identifies the cell to free.

,CELLNAME=*cellname*
The name of the cell to free.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,CELLADDR=*celladdr*
The address of the cell to free.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-byte pointer field.

,INPUT_CPID=*input_cpid*
An optional input parameter that contains the cell pool ID returned on the BUILD request.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,REGS=SAVE
,REGS=USE
A required parameter that indicates how to deal with the registers.

,REGS=SAVE
The request should save and preserve the contents of 64-bit GPRs 2 - 12, starting at offset 40 in a 144-byte area pointed to by register 13.

,REGS=USE
The request can use registers 2 - 12.

Parameters for REQUEST=DELETE

The following parameters are valid when you specify REQUEST=DELETE:

,INPUT_CPID=*input_cpid*
A required input parameter that contains the cell pool ID returned on the BUILD request.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

Optional parameters

The following parameters are optional:

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify (GPR15), (REG15), or (R15), the value is left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify (GPR0), (GPR00), (REG0), (REG00), or (R0), the value is left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12), (GPR0), (GPR00), (REG0), (REG00), or (R0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
This parameter specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

The IARCP64 caller might receive abend code X'DC4'. For detailed abend code information, see *z/OS MVS System Codes*.

Return and reason codes

When the IARCP64 macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro IAXSERVC provides equated symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equated symbol associated with each reason code.

Table 35. Return and Reason Codes for the IARCP64 Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
00	None	**Equate Symbol:** IARCP64Rc_OK **Meaning:** IARCP64 request successful. **Action:** None required. **BUILD** **Meaning:** Cell pool built **Action:** None required. **DELETE** **Meaning:** Cell Pool deleted and storage freed. **Action:** None required. **GET** **Meaning:** Cell from pool obtained. **Action:** None required. **FREE** **Meaning:** Cell returned to the pool. **Action:** None required.
04	None	**Equate Symbol:** IARCP64Rc_Warn **Meaning:** Warning **Action:** Refer to the action provided with the specific reason code.
04	xx0400xx	**Equate Symbol:** IARCP64RsnGetOutOfCells **Meaning:** The request to the IARCP64 GET service specified EXPAND=NO and the current extent is out of cells. **Action:** Either change the request to specify EXPAND=YES or write logic to deal with no cell available.
08	None	**Equate Symbol:** IARCP64Rc_Fail **Meaning:** Service failed due to running out of resources. **Action:** Refer to the action provided with the specific reason code.
08	xx0401xx	**Equate Symbol:** IARCP64RsnMemlimitExhausted **Meaning:** The request to either the IARCP64 BUILD, IARCP64 GET when the pool is being expanded or the IARST64 GET when a new extent is required was not able to obtain private storage due to the address space MEMLIMIT. **Action:** Either raise the MEMLIMIT of the address space or determine if private storage is being consumed excessively somewhere.

Table 35. Return and Reason Codes for the IARCP64 Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Equate Symbol Meaning and Action**
08	xx0402xx	**Equate Symbol:** IARCP64Rsn64BitCommonExhausted **Meaning:** The request to either the IARCP64 BUILD, IARCP64 GET when the pool is being expanded or the IARST64 GET when a new extent is required was not able to obtain common storage due to there being insufficient 64-bit common storage to satisfy the request. **Action:** For common storage, either raise the system limit on common (HVCOMMON) or determine if common storage is being consumed excessively somewhere.

Examples

1. Build a pool according to the following specifications:
   - Cells 32 bytes long
   - In private storage
   - With an owning task of the current task
   - Dumped similar to "RGN" processing
   - Not fetch-protected
   - Pageable storage
   - In key 3
   - Provide a diagnostic trailer.

     Note: Requesting a diagnostic trailer causes the cell size to internally be rounded up 32 - 48 bytes
   - Provide return code if the request is not successful

   The coding sample follows

```
         IARCP64 REQUEST=BUILD,HEADER=theHeader,
               CELLSIZE=theCellsize,OUTPUT_CPID=theCPID,
               COMMON=NO,OWNINGTASK=CURRENT,DUMP=LIKERGN,
               FPROT=NO,TYPE=PAGEABLE,
               CALLERKEY=NO,KEY00TOF0=theKEY,
               TRAILER=YES,FAILMODE=RC,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,
               MF=(E,IARCP64L)

( Place code to check return/reason codes here.)

theHEADER    DC   CL24        Header for pool
theCellsize  DC   F'32'       32-byte cells
Key00ToF0    DC   X'30'       Key 3 (bits 0-3 of the byte)

IAXSERVC                      Return/Reason code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
theCPID  DS    D
IARCP64 MF=(L,IARCP64L)
```

2. Obtain a cell from the pool.
   - Do not expand the pool if no cell is available
   - Provide Return Code if the request is not successful
   - Save and restore registers

   The coding sample follows

```
         IARCP64 REQUEST=GET,INPUT_CPID=theCPID,
               CELLADDR=theCellAddr,
               EXPAND=NO,
               FAILMODE=RC,
```

```
               REGS=SAVE,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,

(Place code to check return/reason codes here.)

IAXSERVC              Return/Reason code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
theCPID  DS    D
theCellAddr DS D
```

3. Free a cell.

   - Save and restore registers

   The coding sample follows

```
         IARCP64 REQUEST=FREE,
               CELLADDR=theCellAddr,
               REGS=SAVE

IAXSERVC              Return/Reason code information
DYNAREA  DSECT
theCPID  DS    D
theCellAddr DS D
```

4. Delete the pool.

   The coding sample follows

```
         IARCP64 REQUEST=DELETE,INPUT_CPID=theCPID,
               MF=(E,IARCP64L)

IAXSERVC              Return/Reason code information
DYNAREA  DSECT
theCPID  DS    D
IARCP64 MF=(L,IARCP64L)
```

Chapter 30. IARR2V — Convert a central storage address to a virtual storage address

Description

Use the IARR2V macro to convert a central storage address to a virtual storage address. This conversion can be useful when you have the central storage address from handling I/O or doing diagnostic support and need to know the corresponding virtual address.

When the input storage address is a central storage address that backs a single page, the system returns the ASID that indicates the address space that owns the central storage, and the STOKEN that indicates the address space or data space that uses the central storage. When a central storage address does not back any page, or backs a read-only nucleus page, the system returns a non-zero return code and reason code.

For more information on the use of the IARR2V macro, see *z/OS MVS Programming: Authorized Assembler Services Guide*. IARR2V is also described in the *z/OS MVS Programming: Assembler Services Guide* with the exception of the LINKAGE parameter.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key. For the LINKAGE parameter, supervisor state with any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24-, 31- or 64-bit.
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller should not hold the RSM locks or any locks higher in the locking hierarchy.
Control parameters:	None.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IARR2V macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
ASID if return code is 0 or 4; otherwise, reason code. The ASID value is X'FFFF' if the returned virtual address represents common storage.

1
Virtual storage address if return code is 0 or 4; otherwise, used as a work register by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Return code.

When control returns to the caller, the ARs contain:

Register
Contents

0
First four bytes of STOKEN if return code is 0 or 4; otherwise, used as a work register by the system.

1
Last four bytes of STOKEN if return code is 0 or 4; otherwise, used as a work register by the system.

2-13
Unchanged.

14
Total shared view count if return code is 0 or 4; otherwise, used as a work register by the system.

15
Valid shared view count if return code is 0 or 4; otherwise, used as a work register by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IARR2V macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARR2V.
IARR2V	

Syntax	Description
ƀ	One or more blanks must follow IARR2V.
RSA=*rsa_addr*	*rsa_addr*: RS-type address, or register (2) - (12).
RSA64=*rsa_addr64*	*rsa_addr64*: RS-type address, or register (2) - (12).
,VSA=*vsa_addr*	*vsa_addr*: RS-type address, or register (2) - (12).
,VSA64=*vsa_addr64*	*vsa_addr64*: RS-type address, or register (2) - (12).
,ASID=*asid_addr*	*asid_addr*: RS-type address, or register (2) - (12).
,STOKEN=*stoken_addr*	*stoken_addr*: RS-type address, or register (2) - (12).
,WORKREG=*work_reg*	*work_reg*: RS-type address, or register (2) - (12).
,WORKREG=NONE	**Default**: WORKREG=NONE
,NUMVIEW=*view_addr*	*view_addr*: RS-type address, or register (2) - (12).
,NUMVALID=*val_addr*	*val_addr*: RS-type address, or register (2) - (12).
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode*	*retcode*: RS-type address, or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address, or register (2) - (12).

Parameters

The parameters are explained as follows:

RSA=*rsa_addr*
Specifies the name (RS-type) or address (in register 2-12) of an input fullword that contains the central storage address to be converted to a virtual storage address. This keyword is used to provide a 31–bit real address. RSA and RSA64 are mutually exclusive keywords. You must specify one or the other.

RSA64=*rsa_addr64*
Specifies the name (RS-type) or address (in register 2-12) of an input doubleword that contains the central storage address to be converted to a virtual storage address. This keyword is used to provide a 64–bit real address. RSA and RSA64 are mutually exclusive keywords. You must specify one or the other. To use this keyword, the SYSTATE macro must be invoked specifying ARCHLVL greater than 1.

,VSA=*vsa_addr*
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the virtual storage address that corresponds to the input central storage address.

,VSA64=*vsa_addr64*
Specifies the name (RS-type) or address (in register 2-12) of an optional output doubleword that the system uses to return the 64–bit virtual storage address that corresponds to the input central storage address. VSA and VSA64 are mutually exclusive keywords. To use this keyword, the SYSTATE macro must be invoked specifying ARCHLVL greater than 1.

,ASID=*asid_addr*
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the ASID of the address space associated with the output virtual storage address. The system returns the ASID in bits 16-31 of the fullword, and clears bits 1-15 to 0. If the input central storage address backs a page that is shared through the use of the IARVSERV macro, the system sets bit 0 to 1; otherwise, bit 0 contains 0.

,STOKEN=*stoken_addr*
Specifies the name (RS-type) or address (in register 2-12) of an optional 8-character output field that the system uses to return the STOKEN for the address space or data space associated with the output virtual storage address.

,WORKREG=*work_reg*
,WORKREG=NONE
Specifies whether the system is to return a page sharing view count. If you want the system to return a page sharing view count, specify *work-reg* as a digit from 2 through 12 that identifies a GPR/AR pair that the system can use as work registers. WORKREG=*work_reg* is required if you code NUMVIEW or NUMVALID.

WORKREG=NONE is the default and specifies that the system is not to return the sharing count.

,NUMVIEW=*view_addr*
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the number of page sharing views associated with the input central storage address. This number is non-zero only if the system sets bit 0 of the ASID. NUMVIEW=*view_addr* is required with the WORKREG=*work_reg* parameter.

,NUMVALID=*val_addr*
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword that the system uses to return the number of valid page sharing views associated with the input central storage address. A valid page must be currently defined in central storage. This number is non-zero only if the system sets bit 0 of the *asid_addr*. NUMVALID=*val_addr* is required with the WORKREG=*work_reg* parameter.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Specifies whether the system is to use a program call (LINKAGE=SYSTEM) or branch entry (LINKAGE=BRANCH). LINKAGE=SYSTEM is the default.

,RETCODE=*retcode*
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode*
Specifies the name (RS-type) or address (in register 2-12) of an optional output fullword into which the system copies the a reason code from GPR 0.

ABEND codes

None.

Return and reason codes

When the IARR2V macro returns control to your program, GPR 15 (and *retcode* if you coded RETCODE) contains the return code. If the return code is not 0 or 4, GPR 0 (and *rsncode* if you coded RSNCODE) contains the reason code.

Table 36. Return and Reason Codes for the IARR2V Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: The IARR2V request completed successfully. The address returned in the VSA parameter represents an address space page. **Action**: None required.
04	None	**Meaning**: The IARR2V request completed successfully. The address returned in the VSA parameter represents a data space page. **Action**: None required.
08	xx0001xx	**Meaning**: Program error. The IARR2V request was unsuccessful because the input central storage address was not within the bounds of central storage. **Action**: Check your input central storage address and rerun the program.
08	xx0002xx	**Meaning**: Program error. The IARR2V request was unsuccessful because the frame corresponding to the input central storage address was not assigned to a page. **Action**: Check your input central storage address and rerun the program.
08	xx0003xx	**Meaning**: Program error. The IARR2V request was unsuccessful because the frame corresponding to the input central storage address contains shared data, but no virtual address for any accessible address space (either home, primary, or secondary) corresponds to the frame. **Action**: Check your input central storage address and rerun the program.
08	xx0004xx	**Meaning**: System error. The IARR2V request was recursively invoked. **Action**: Record the return code and reason code and supply them to the appropriate IBM support personnel.
08	xx0005xx	**Meaning**: Program error. The IARR2V request was unsuccessful because the frame corresponding to the input central storage address was assigned, but the data space STOKEN could not be found. **Action**: Check your input central storage address and rerun the program.
08	xx0006xx	**Meaning**: Program error. The IARR2V request was unsuccessful because the virtual address is above 2G and the caller did not specify VSA64. **Action**: Specify VSA64 on the IARR2V invocation.
08	xx0007xx	**Meaning**: The frame corresponding to the input real storage is used to back a high virtual shared page but the address space does not have access to it (none of its home, primary, or secondary address space did the ShareMemObj) **Action**: None.

Example 1

Convert the central storage address in variable VSA and place the result in variable VSAOUT.

```
         LRA   1,VSA
         LR    5,1
INVOKE1  IARR2V RSA=(5),VSA=VSAOUT
  .
  .
VSA      DS    F
VSAOUT   DS    F
```

Example 2

Same as Example 1, but return ASID in variable ASIDO.

```
INVOKE2  IARR2V RSA=(5),ASID=ASIDO
  .
  .
ASIDO    DS    F
```

Example 3

Same as Example 1, but return STOKEN in variable STOKO.

```
INVOKE3  IARR2V RSA=(5),STOKEN=STOKO
  .
  .
STOKO    DS    F
```

Example 4

Obtain the total and valid number of page sharing views associated with the input address. WORKREG is required.

```
INVOKE4  IARR2V RSA=(5),WORKREG=(6) ,NUMVIEW=VIEWS, NUMVALID=VALS
  .
  .
VIEWS    DS    F
VALS     DS    F
```

Chapter 31. IARST64 — 64-bit storage services

Description

Use IARST64 to request 64-bit Storage Services.

With IARST64, you can request services to:

- Obtain storage (REQUEST=GET)
- Return storage (REQUEST=FREE)

Note: There is diagnostic support for 64-bit cell pools, which are created by IARST64, in IPCS using the CBFORMAT command. To locate the cell pool of interest, you need to follow the pointers from HP1, to HP2, to the CPHD. For common storage, the HP1 is located in the ECVT. CBF ECVT formats the ECVT, then does a FIND on HP1. Extract the address of the HP1 from the ECVT and CBF addrhp1 STR(IAXHP1) formats the HP1. Each entry in the HP1 represents an attribute set (storage key, storage type(pageable, DREF, FIXED), and Fetch-Protection (ON or OFF)). The output from this command contains CBF commands for any connected HP2s. Select the CBF command of interest and run it to format the HP2. The HP2 consists of pointers to cell pool headers for different sizes. Choose the size of interest and select the command that looks like this to format the cell pool header:

```
CBF addrchphd STR(IAXCPHD)
```

To see details about all of the cells in the pool, use the EXIT option as follows:

```
CBF addrcphd STR(IAXCPHD) EXIT
```

For private storage, the HP1 is anchored in the STCB. The quickest way to locate the HP1 is to run the SUMMARY FORMAT command for the address space of interest. Locate the TCB that owns the storage of interest and then scroll down to the formatted STCB. The HP1 field contains the address of the HP1. From here, the processing is the same as described for common storage.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Use of the COMMON=YES, TYPE=DREF, TYPE=FIXED, OWNINGTASK=RCT, or the Key00ToF0 parameter with a value other than 9 requires the caller to be running in key 0-7. Use of MEMLIMIT=NO requires key 0-7 or supervisor state. All other options have a minimum authorization of problem state and PSW key 8-15.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	• The caller can be enabled or disabled for interrupts when requesting storage that is defined as COMMON=YES and TYPE=DREF or TYPE=FIXED. • For all other parameter combinations, the caller must be enabled for interrupts.

Environmental factor	Requirement
Locks:	For the GET request, the following locks can be held by the caller or must be obtainable by IARST64: • For requests with COMMON=NO, the locking restrictions are the same as for IARV64 REQUEST=GETSTOR. For the FREE request, the caller might hold locks but is not required to hold any.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

None.

Input register information

When REGS=SAVE is not used, the caller does not have to place any information into any general-purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IARST64 macro with REGS=SAVE, the caller must ensure that the following GPR contains the specified information:

Register
Contents

13
Address of a 144-byte area within which the 88 bytes beginning at offset 40 can be modified.

Before issuing the IARST64 macro, the caller does not have to place any information into any access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the 64-bit GPRs contain:

For REQUEST=GET

Register
Contents

0
Reason code in the low 32 bits if the return code is not 0. Otherwise, used as a work register by the system.

1
The address of the obtained storage.

2-12
Unchanged if REGS=SAVE was specified, used as work registers by the system if REGS=USE was specified.

13
Unchanged.

14
Used as a work register by the system.

15
Return code in the low 32 bits.

For REQUEST=FREE

Register
Contents

0-1
Used as a work register by the system.

2-12

- Unchanged, if REGS=SAVE was specified.
- Used as work registers by the system, if REGS=USE was specified.

13
Unchanged.

14-15
Used as a work register by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

Some callers depend on register contents to remain the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IARST64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARST64.
IARST64	
␣	One or more blanks must follow IARST64.
REQUEST=GET	

Syntax	Description
REQUEST=FREE	
,SIZE=*size*	*size:* RS-type address or address in register (2) - (12)
,AREAADDR=*areaaddr*	*areaaddr:* RS-type address or address in register (2) - (12)
,COMMON=NO	
,COMMON=YES	
,OWNINGTASK=CURRENT	
,OWNINGTASK=MOTHER	
,OWNINGTASK=IPT	
,OWNINGTASK=JOBSTEP	
,OWNINGTASK=CMRO	
,OWNINGTASK=RCT	
,MEMLIMIT=YES	**Default:** MEMLIMIT=YES
,MEMLIMIT=NO	
,LOCALSYSAREA=NO	**Default:** LOCALSYSAREA=NO
,LOCALSYSAREA=YES	
,OWNER=HOME	
,OWNER=PRIMARY	
,OWNER=SECONDARY	
,OWNER=SYSTEM	
,OWNER=BYASID	
,OWNINGASID=*owningasid*	*owningasid:* RS-type address or address in register (2) - (12)
,FPROT=YES	
,FPROT=NO	
,SENSITIVE=UNKNOWN	**Default:** SENSITIVE=UNKNOWN
,SENSITIVE=YES	

Syntax	Description
,SENSITIVE=NO	
,TYPE=PAGEABLE	
,TYPE=DREF	
,TYPE=FIXED	
,CALLERKEY=YES	
,CALLERKEY=NO	
,KEY00TOF0=*key00tof0*	*key00tof0:* RS-type address or address in register (2) - (12)
,FAILMODE=RC	
,FAILMODE=ABEND	
,REGS=SAVE	
,REGS=USE	
,AREANAME=*areaname*	*areaname:* RS-type address or address in register (2) - (12)
,AREAADDR=*areaaddr*	*areaaddr:* RS-type address or address in register (2) - (12)
,REGS=SAVE	
,REGS=USE	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12), (GPR15), (REG15), or (R15).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12), (GPR0), (GPR00), (REG0), (REG00), or (R0).

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1 that is the name on the IARST64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=GET
REQUEST=FREE
A required parameter that indicates the type of request.

REQUEST=GET
This parameter gets storage.

REQUEST=FREE
This parameter returns storage.

Note:

This request is unconditional, and will abnormally end if there is a problem. No return and reason codes are provided, so do not specify the RETCODE and RSNCODE parameters.

,SIZE=*size*
When REQUEST=GET is specified, a required input parameter that indicates the size of the storage to be obtained. The size can be anywhere 1 - 128 K bytes. The size is rounded up to the power of 2. So cell sizes are 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16,384, 32,768, 65,536 and 131,072 bytes. The smallest cell size that contains the request is used. If the requested size is at least 4 bytes less than the rounded up cell size, a trailer will be added to check for storage overruns. For storage that is larger than what IARST64 supports, consider using IARCP64 or IARV64 GETSTOR or GETCOMMON. Do not specify a value that exceeds 128 K or incorrect results can ensue.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,AREAADDR=*areaaddr*
When REQUEST=GET is specified, an optional output parameter, of the obtained storage. If AREAADDR is not specified, the cell address is left in register 1.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-byte pointer field.

,COMMON=NO
,COMMON=YES
When REQUEST=GET is specified, a required parameter that indicates if the pool is to reside in common storage.

,COMMON=NO
This parameter indicates that the pool is not to reside in common storage.

,COMMON=YES
This parameter indicates that the pool is to reside in common storage.

,OWNINGTASK=CURRENT
,OWNINGTASK=MOTHER
,OWNINGTASK=IPT
,OWNINGTASK=JOBSTEP
,OWNINGTASK=CMRO
,OWNINGTASK=RCT
When COMMON=NO and REQUEST=GET are specified, a required parameter that indicates the task that is to be considered the owner.

,OWNINGTASK=CURRENT
This parameter indicates that the current task is to be the owner. Do not specify this parameter unless the program is in task mode.

,OWNINGTASK=MOTHER
This parameter indicates that the mother task of the current task is to be the owner. If the current task is the cross-memory resource owning task, the request will fail. Do not specify this parameter unless the program is in task mode.

,OWNINGTASK=IPT
This parameter indicates that the initial pthread task (subtask running under UNIX System Services) is to be the owner. If the current task or mother task is not the IPT, then this will default to the current task as the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=JOBSTEP
This parameter indicates that the jobstep task of the current task (the task with TCB address in field TCBJSTCB of the current task's TCB) is to be the owner. Do not specify this unless the program is in task mode.

,OWNINGTASK=CMRO
This parameter indicates that the cross-memory resource-owning task is to be the owner.

,OWNINGTASK=RCT
This parameter indicates that the region control task (RCT) is to be the owner. You must be key 0-7 to request this option.

,MEMLIMIT=YES
,MEMLIMIT=NO
When COMMON=NO and REQUEST=GET are specified, an optional parameter that indicates whether MEMLIMIT applies if an additional 1 M segment is obtained to satisfy the request. The default is MEMLIMIT=YES.

,MEMLIMIT=YES
This parameter indicates that MEMLIMIT applies.

,MEMLIMIT=NO
This parameter indicates that MEMLIMIT does not apply.

,LOCALSYSAREA=NO
,LOCALSYSAREA=YES
When Common=NO and request=GET are specified, an optional parameter that specifies whether this is an explicit allocation request for 64-bit virtual storage in the local system area. The LOCALSYSAREA parameter can be used only by callers running in supervisor state or with a PSW key 0-7. THE DEFAULT IS LOCALSYSAREA=NO.

,LOCALSYSAREA=NO
The request will not be satisfied from the local system area.

,LOCALSYSAREA=YES
The request is to be satisfied from the local system area. The storage that is obtained with this keyword will not be copied during Fork processing. The use of local system area storage does not preclude checkpoint or restart from succeeding.

,OWNER=HOME
,OWNER=PRIMARY
,OWNER=SECONDARY
,OWNER=SYSTEM
,OWNER=BYASID
When COMMON=YES and REQUEST=GET are specified, a required parameter that designates the owner of the storage.

,OWNER=HOME
This parameter indicates that the home address space is to be the owner.

,OWNER=PRIMARY
This parameter indicates that the primary address space is to be the owner.

,OWNER=SECONDARY
This parameter indicates that the secondary address space is to be the owner.

,OWNER=SYSTEM
This parameter indicates that the system is to be the owner. Use this only when there is no specific address space, which can be considered the owner.

,OWNER=BYASID
This parameter indicates that the owner is the ASID specified by the OwningASID parameter.

,OWNINGASID=*owningasid*
When OWNER=BYASID, COMMON=YES and REQUEST=GET are specified, a required input parameter that specifies the ASID that is to be the owner. A value of 0 is equivalent to having specified OWNER=SYSTEM. Do not specify a value which exceeds 32767 or incorrect results can ensue.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,FPROT=YES
,FPROT=NO
When REQUEST=GET is specified, a required parameter that indicates if the pool storage is to be fetch-protected.

,FPROT=YES
This parameter indicates that the pool storage is to be fetch-protected.

,FPROT=NO
This parameter indicates that the pool storage is not to be fetch-protected.

,SENSITIVE=UNKNOWN
,SENSITIVE=YES
,SENSITIVE=NO
When REQUEST=GET is specified, an optional keyword input that specifies whether the memory will contain data that is potentially sensitive (for instance, personal or confidential). The default is SENSITIVE=UNKNOWN.

,SENSITIVE=UNKNOWN
Specifies that the sensitivity of the data is not known. The memory may or may not contain sensitive data.

,SENSITIVE=YES
Specifies that the memory contains sensitive data.

,SENSITIVE=NO
Specifies that the memory does not contain sensitive data.

,TYPE=PAGEABLE
,TYPE=DREF
,TYPE=FIXED
When REQUEST=GET is specified, a required parameter that indicates the type of storage for the pool.

,TYPE=PAGEABLE
This parameter indicates that the pool storage is to be pageable.

,TYPE=DREF
This parameter indicates that the pool storage is to be disabled-reference (DREF).

,TYPE=FIXED
This parameter indicates that the pool storage is to be page-fixed.

,CALLERKEY=YES
,CALLERKEY=NO
When REQUEST=GET is specified, a required parameter that indicates if the pool storage is to be in the key of the caller of the GET request.

,CALLERKEY=YES
This parameter indicates that the pool storage is to be in the key of the caller.

,CALLERKEY=NO
This parameter indicates that the pool storage is not to be in the key of the caller, but instead in the key specified by the Key00ToF0 parameter.

,KEY00TOF0=*key00tof0*
When CALLERKEY=NO and REQUEST=GET are specified, a required input parameter that indicates the key for the pool storage. The value should be in the range X'00' to X'70' (that is, the key 0 - 7 in the high-order 4 bits of the byte) for a caller that is key 0. For callers in key 1 - 7, there is no reason to use this parameter. The value X'90' is the only accepted key for a caller that is key 8 - 15. Be sure that the value is a multiple of 16 within the required range or incorrect results can occur.

To code: Specify the RS-type address, or address in register (2) - (12), of a one-byte field.

,FAILMODE=RC
,FAILMODE=ABEND
When REQUEST=GET is specified, a required parameter that indicates what to do if the GET request is not successful due to out of memory in the requested area conditions.

,FAILMODE=RC
This parameter returns with a failure return code.

Note: There are cases for which an ABEND occurs regardless of the specification of FAILMODE=RC.

,FAILMODE=ABEND
This parameter abnormally ends.

,REGS=SAVE
,REGS=USE
When REQUEST=GET is specified, a required parameter that indicates how to deal with the registers.

,REGS=SAVE
This parameter saves and preserves the contents of 64-bit GPRs 2 - 12 starting at offset 40 in a 144-byte area pointed to by register 13.

,REGS=USE
This parameter indicates that the system uses registers 2 - 12 as work registers and does not restore them.

,AREANAME=*areaname*
,AREAADDR=*areaaddr*
When REQUEST=FREE is specified, a required input parameter.

,AREANAME=*areaname*
A parameter that is the area to free.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,AREAADDR=*areaaddr*
A parameter that contains the address of the area to free.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-byte pointer field.

,REGS=SAVE
,REGS=USE
When REQUEST=FREE is specified, a required parameter that indicates how to deal with the registers.

,REGS=SAVE
This parameter saves and preserves the contents of 64-bit GPRs 2 - 12 starting at offset 40 in a 144-byte area pointed to by register 13.

,REGS=USE
This parameter indicates that you can use registers 2 - 12.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, or R15 (within or without parentheses), the value will be left in GPR15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (0), (GPR0), (GPR00), (REG0), (REG00), or (R0).

ABEND codes

The IARST64 caller might receive abend code X'DC4'. For detailed abend code information, see *z/OS MVS System Codes*.

In the following IARST64 abend reason codes, the bytes designated "xx" are for diagnostic purposes and have no significance to the external interface. Equate IARST64AbendRsncodeMask has been provided to let you build a mask to ignore those bytes.

Hexadecimal Reason Code	Equate Symbol Meaning and Action
xx0410xx	**Equate Symbol:** IARST64AbendRsnCellAddrLow **Meaning:** The storage address passed to the IARST64 FREE service is within a megabyte used for storage pools, but the address is less than the address of the first usable storage address. **Action:** Correct the address passed to IARST64 FREE, making sure it is the same address that was returned from IARST64 GET.
xx0413xx	**Equate Symbol:** IARST64AbendRsnCellNotInExtent **Meaning:** The request was to the IARCP64 or IARST64 FREE service and the address of the storage passed in, is not within the bounds of a cell pool. **Action:** The address passed to IARST64 REQUEST=FREE must be the same as the address obtained from IARST64 REQUEST=GET.
xx0419xx	**Equate Symbol:** IARST64AbendRsnCellOverRun **Meaning:** The request was to the IARCP64 or IARST64 FREE service and the trailer data at the end of the cell was detected as being overrun. If the overrun is sufficiently large, it will cause damage to the following cell. The caller is abnormally ended so they can fix the code to not use more storage than requested. **Action:** Determine whether the storage has been overrun or whether the trailer data was overlaid by some other code. Fix the code so it only uses the amount of storage requested.
xx041Axx	**Equate Symbol:** IARST64AbendRsnCellNotInUse **Meaning:** The request was to the IARCP64 or IARST64 FREE service and the address of the storage passed in, is already in the freed state. This will happen when an application frees the storage twice. **Action:** Determine whether the current application is freeing the storage twice or whether it is using a cell that some other storage is freeing twice.
xx041Bxx	**Equate Symbol:** IARST64AbendRsnNotOnCellBoundary **Meaning:** The request was to the IARCP64 or IARST64 FREE service and the address of the storage passed in is not on a cell boundary in the cell pool from which the GET request was satisfied. **Action:** When freeing storage with IARST64 REQUEST=FREE, make sure to specify the address that was returned by IARST64 REQUEST=GET.
xx041Cxx	**Equate Symbol:** IARST64AbendRsnIARV64Error **Meaning:** During processing of IARST64 GET, a call to the IARV64 service for GETSTOR, GETCOMMON, PAGEFIX, or PROTECT failed. The failing return code from IARV64 was placed in register 2 before the abend. The failing reason code from IARV64 was placed in register 3 before the abend. **Action:** Examine the return and reason code as documented under IARV64 to determine if the problem is one that you can resolve.

Hexadecimal Reason Code	Equate Symbol Meaning and Action
xx0420xx	**Equate Symbol:** IARST64AbendRsnCphaNotQueue **Meaning:** The cell pool header authorized area was not queued to the owning task as expected. This could happen due to storage overlays or the caller bypassing the IARST64 macro and PCing directly to the service with incorrect input parameters. **Action:** Make sure that the application is using the IARST64 macro to request storage. If the problem persists, collect a dump and contact IBM Service.
xx0425xx	**Equate Symbol:** IARST64AbendRsnPoolNotInCallerKey **Meaning:** The request to IARST64 GET was against a storage pool that was not in the key of the caller. Normally this will abend with an 0C4, but if the pool is out of cells and is in storage that is not fetch-protected, the pool expand routine verifies that the caller can modify this storage pool. **Action:** You must be in a key that has the ability to modify the pool storage for the request to be processed.
xx0426xx	**Equate Symbol:** IARST64AbendRsnPrimaryExtentOverlaid **Meaning:** The request to IARST64 or IARCP64 GET was against a storage pool where the primary extent control information has been overlaid. **Action:** Collect a dump and report the problem to IBM.
xx0427xx	**Equate Symbol:** IARST64AbendRsnSecondaryExtentOverlaid **Meaning:** The request to IARST64 or IARCP64 GET was against a storage pool where the secondary extent control information has been overlaid. **Action:** Collect a dump and report the problem to IBM.
xx0428xx	**Equate Symbol:** IARST64AbendRsnUnexpectedError **Meaning:** During processing of IARST64 GET an unexpected abend occurred. An SDUMP should have been generated. **Action:** Collect the dump and report the problem to IBM.
xx0511xx	**Equate Symbol:** IARST64AbendRsnKeyGT7Common **Meaning:** The request to IARST64 GET was for common storage, but the requested or caller was greater than key 7. You cannot allocate common storage in key 8 or above. **Action:** Correct the key passed to IARST64 GET or change your request to get private storage.
xx0512xx	**Equate Symbol:** IARST64AbendRsnGetMotherFromCmro **Meaning:** The request was to the IARST64 GET service and specified OWNINGTASK(MOTHER), but the caller is running on the CMRO task. You can't request the mother task be the storage owner from the CMRO task. **Action:** Either specify CMRO as the owner or specify RCT if you want the storage to persist across termination of the CMRO.
xx0514xx	**Equate Symbol:** IARST64AbendRsnGetNotRctOrCmro **Meaning:** The request was to the IARST64 GET service for private storage and the caller was running in cross memory mode or SRB mode. In these environments, the OWNINGTASK parameter must be set to RCT or CMRO. Neither of these were specified, so the request is failed. **Action:** Specify the OWNINGTASK parameter as RCT or CMRO.

Hexadecimal Reason Code	Equate Symbol Meaning and Action
xx0515xx	**Equate Symbol:** IARST64AbendRsnGetCellSizeZero **Meaning:** The request was to the IARST64 GET service and specified a length of zero. **Action:** Specify a length between 1 and 128 K.
xx0516xx	**Equate Symbol:** IARST64AbendRsnGetNotAuth **Meaning:** The request was to the IARST64 GET service and specified a parameter that requires the caller to be running in key 0-7. The caller is not authorized to use authorized options of COMMON, DREF, FIXED, OWNINGTASK(RCT), CALLERKEY(NO) and Key00ToF0 set to a system key. **Action:** Either run the code in key 0-7 or do not use authorized options.
xx0517xx	**Equate Symbol:** IARST64AbendRsnGetCellSizeTooBig **Meaning:** The request was to the IARST64 GET service and specified a length greater than 128 K. **Action:** Specify a size between 1 and 128 K. If larger storage is needed, consider using IARCP64 or IARV64 GETSTOR or GETCOMMON.
xx0518xx	**Equate Symbol:** IARST64AbendRsnGetKeyNot9 **Meaning:** The request was to the IARST64 GET service and specified a CALLERKEY(NO) and a value for Key00ToF0 that was not key 9 and the caller is not authorized. **Action:** The only key that an unauthorized user can specify is key 9. Either request key 9 or change the specification to CALLERKEY(YES).
xx0529xx	**Equate Symbol:** IARST64AbendRsnGetSizeTooBig **Meaning:** The call to the IARST64 GET service specified a cell size larger than the maximum size supported. **Action:** Specify a size between 1 and 128 K. If a larger storage area is needed, consider using IARCP64 or IARV64 REQUEST=GETSTOR or GETCOMMON.
xx052Axx	**Equate Symbol:** IARST64AbendRsnValidationError **Meaning:** The call to the IARST64 GET service detected a validation error when locating the storage pool to be used. Possible cause is storage overlay of the storage pool control block in the caller's key. **Action:** Collect a dump and report the problem to IBM.
xx052Bxx	**Equate Symbol:** IARST64AbendRsnMemLimitNoUnauth **Meaning:** The call to the IARST64 GET service requested MEMLIMIT=NO, but is running unauthorized (key 8-15 and problem program state). **Action:** Either specify MEMLIMIT=YES or call from an authorized environment.
xx052Cxx	**Equate Symbol:** IARST64AbendRsnCellLT4Gig **Meaning:** The call to the IARCP64 or IARST64 FREE service was passed a cell address less than 4 Gig, so it can't possibly be a valid cell address in a 64 bit cell pool. **Action:** Only pass a storage address that was obtained with IARCP64 or IARST64 GET.
xx052Dxx	**Equate Symbol:** IARST64AbendRsnLocalSysAreaYesUnauth **Meaning:** The call to the IARST64 GET service requested LOCALSYSAREA=YES, but is running unauthorized (key 8-15 and problem program state). **Action:** Either specify LOCALSYSAREA=NO or CALL from an authorized environment.

Hexadecimal Reason Code	Equate Symbol Meaning and Action
xx052Exx	**Equate Symbol:** IARST64AbendRsnKeyGT7 **Meaning:** The call to the IARST64 GET service requested private storage with CALLERKEY (NO) and KEY00TOF0 greater than key 7 when the CALLERKEY was less than key 8. An authorized caller cannot request unauthorized storage. **Action:** Correct the key of the requested storage or change the caller key.

Return and reason codes

When the IARST64 macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro IAXSERVC provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

Table 37. Return and Reason Codes for the IARST64 Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
00	None	**Equate Symbol:** IARST64Rc_OK **Meaning:** IARST64 request successful. **Action:** None required. **GET** **Meaning:** storage obtained of requested size and attributes **Action:** None required. **FREE** **Meaning:** storage freed **Action:** None required.
08	None	**Equate Symbol:** IARST64Rc_Fail **Meaning:** Service failed due to running out of resources. **Action:** Refer to the action provided with the specific reason code.
08	xx0401xx	**Equate Symbol:** IARST64RsnMemlimitExhausted **Meaning:** The request to the IARST64 GET service was not able to obtain storage due to address space limits. **Action:** Either raise the MEMLIMIT of the address space or determine if private storage is being consumed excessively somewhere.
08	xx0402xx	**Equate Symbol:** IARST64Rsn64BitCommon Exhausted **Meaning:** The request to the IARST64 GET service was not able to obtain storage due to system limits. **Action:** For common storage, either raise the system limit on common (HVCOMMON) or determine if common storage is being consumed excessively somewhere.
08	xx0403xx	**Equate Symbol:** IARST64RsnMemlimitZero **Meaning:** The request to IARST64 GET was not able to obtain private storage due to the address space MEMLIMIT being set to zero. **Action:** Either set the MEMLIMIT of the address space to a nonzero value or if authorized, specify MEMLIMIT=NO on the IARST64 GET call to tell the service to bypass the address space MEMLIMIt.

Examples

Example 1

Obtain storage.

Operations:

- 32-byte area
- In private storage
- With an owning task of the current task
- Dumped similar to "LSQA" processing (triggered by DREF or FIXED)
- Fetch-protected
- DREF storage
- In Key 7
- Provide Return Code if the request is not successful
- Save and restore registers

The code is as follows:

```
IARST64 REQUEST=GET,
    AREAADDR=theAreaAddr,
    SIZE=theAreaSize,
    COMMON=NO,OWNINGTASK=CURRENT,
    FPROT=YES,TYPE=DREF,
    CALLERKEY=NO,KEY00TOF0=theKEY,
    FAILMODE=RC,
    REGS=SAVE,
    RETCODE=LRETCODE,RSNCODE=LRSNCODE,

(Place code to check return code or reason codes here.)
    theAreaSize   DC F‘32’
    theKey      DC  X'70'
    IAXSERVC
    DYNAREA    DSECT
    LRETCODE   DS    F
    LRSNCODE   DS    F
    theAreaAddr   DS D
```

Example 2

Free the storage.

Operation: Save and restore registers.

The code is as follows:

```
IARST64 REQUEST=FREE,
    AREAADDR=theAreaAddr,
    REGS=SAVE,

(There is no return code or reason code from
 IARST64 REQUEST=FREE.)

    IAXSERVC
    DYNAREA    DSECT
    LRETCODE  DS    F
    LRSNCODE  DS    F
    theAreaAddr   DS D
```

Chapter 32. IARSUBSP — Create and delete a subspace

Description

Use the IARSUBSP macro to create and delete subspaces. A subspace is a section of address space private area storage that you have set up to contain and protect a program and its data. Subspaces provide isolation between multiple programs running in a single address space by allowing a program that runs in the subspace to reference only certain storage in the address space private area. For more information about subspaces and how to use them, see *z/OS MVS Programming: Extended Addressability Guide*.

Use the IARSUBSP macro to:

- Identify storage to be assigned to a subspace (IDENTIFY parameter)
- Create a subspace (CREATE parameter)
- Assign the identified storage to the created subspace (ASSIGN parameter)
- Disassociate the identified storage from the created subspace (UNASSIGN parameter)
- Delete a subspace (DELETE parameter)
- Make the storage ineligible to be assigned to a subspace (UNIDENTIFY parameter).

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For the ASSIGN and UNASSIGN parameters, problem state with any PSW key. For IDENTIFY, CREATE, DELETE, and UNIDENTIFY, supervisor state or PSW key 0 - 7.
Dispatchable unit mode:	For IDENTIFY, CREATE, DELETE, and UNIDENTIFY, task. For ASSIGN and UNASSIGN, task or SRB.
Cross memory mode:	For IDENTIFY, CREATE, DELETE, and UNIDENTIFY, PASN=HASN=SASN. For ASSIGN and UNASSIGN, PASN=HASN, any SASN.
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller cannot hold locks.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

Before issuing IARSUBSP, the caller must obtain storage for the subspace by using the STORAGE or GETMAIN macro. See the RANGLIST parameter description for the required attributes of this storage. The caller must not release this storage until after issuing IARSUBSP UNIDENTIFY.

Restrictions

None.

Input register information

Before issuing the IARSUBSP macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
A reason code, if GPR 15 contains a non-zero return code; otherwise, used as a work register by the system.

1
Used as a work register by the system.

2 - 13
Unchanged.

14
Used as a work register by the system.

15
A return code.

When control returns to the caller, the ARs contain:

Register
Contents

0- 1
Used as a work register by the system.

2 - 13
Unchanged.

14 - 15
Used as a work register by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IARSUBSP macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARSUBSP.

Syntax	Description
IARSUBSP	
ƀ	One or more blanks must follow IARSUBSP.
	Valid parameters (required parameters are underlined):
IDENTIFY	RANGLIST,NUMRANGE
CREATE	NAME,STOKEN,GENNAME,OUTNAME
ASSIGN	RANGLIST,STOKEN,NUMRANGE
UNASSIGN	RANGLIST,STOKEN,NUMRANGE
DELETE	STOKEN
UNIDENTIFY	RANGLIST,NUMRANGE
,RANGLIST=*ranglist_addr*	RS-type address, or address in register (2) - (12).
,NUMRANGE=*numrange_addr*	RS-type address, or address in register (2) - (12).
	Default: 1 range
,NAME=*name_addr*	RS-type address, or address in register (2) - (12).
,GENNAME=NO	**Default**: GENNAME=NO
,GENNAME=COND	
,GENNAME=YES	
,OUTNAME=*outname_addr*	RS-type address, or address in register (2) - (12).
,STOKEN=*stoken_addr*	RS-type address, or address in register (2) - (12).

Parameters

The IDENTIFY, CREATE, ASSIGN, UNASSIGN, DELETE, and UNIDENTIFY parameters designate the services of the IARSUBSP macro, and are mutually exclusive.

The parameters are explained as follows:

IDENTIFY
Identifies the ranges of storage specified on the RANGLIST parameter as eligible to be assigned to a subspace. When the IDENTIFY function successfully completes, the storage specified on the RANGLIST parameter cannot be referenced by a program running in a subspace until that storage is assigned to that subspace.

When you issue the IARSUBSP macro with IDENTIFY, you must specify the RANGLIST parameter. The NUMRANGE parameter is optional.

CREATE
Requests that the system create a subspace, and return an STOKEN by which a program can identify the subspace.

When you issue the IARSUBSP macro with CREATE, the NAME and STOKEN parameters are required. The GENNAME and OUTNAME parameters are optional.

ASSIGN
Requests that the system associate the range of storage specified on the RANGLIST parameter with the subspace indicated by the STOKEN parameter. When the range of storage has been assigned to the subspace, a program can reference the storage by issuing the BSG instruction.

When you issue the IARSUBSP macro with ASSIGN, you must specify the STOKEN and RANGLIST parameters. The NUMRANGE parameter is optional.

UNASSIGN
Requests that the system disassociate the storage identified by the RANGLIST parameter from the subspace identified by the STOKEN parameter. When the request is complete, the range of storage cannot be referenced by a program running in a subspace.

When you issue the IARSUBSP macro with UNASSIGN, you must specify the STOKEN and RANGLIST parameters. The NUMRANGE parameter is optional.

DELETE
Requests that the system delete the subspace indicated by the STOKEN parameter. The subspace can be deleted only by the task that created it.

When you issue the IARSUBSP macro with DELETE, you must specify the STOKEN parameter. Do not code any other parameters.

UNIDENTIFY
Identifies the ranges of storage specified on the RANGLIST parameter as ineligible to be assigned to a subspace.

If a range of storage specified on the RANGLIST parameter is still assigned to a subspace, the system will perform the UNASSIGN function before performing the UNIDENTIFY function.

When you issue the IARSUBSP macro with the UNIDENTIFY parameter, you must specify the RANGLIST parameter. The NUMRANGE parameter is optional.

,RANGLIST=*ranglist_addr*
Specifies the address of a fullword input variable containing the address of the range list. The range list is a list of 8-byte entries in contiguous storage that indicate the ranges of storage to be:

- Made eligible or ineligible to be assigned to a subspace, when specified with the IDENTIFY or UNIDENTIFY functions
- Associated with or disassociated from a subspace, when specified with the ASSIGN or UNASSIGN functions.

Each entry in the range list is 2 fullwords long. The first fullword contains the address of the beginning of the range of storage. The second fullword contains the number of 4-kilobyte (4096 bytes) pages that comprise the range of storage.

When RANGLIST is specified with the IDENTIFY or UNIDENTIFY parameter, the address in the first fullword must begin on a segment boundary. A segment is 1 megabyte (1,048,576 bytes) long. The value of the second fullword must be a multiple of 256.

When RANGLIST is specified with the ASSIGN or UNASSIGN parameters and the storage specified is above 16 megabytes, the requirements for the range list entries are the same as when RANGLIST is specified with IDENTIFY or UNIDENTIFY.

When RANGLIST is specified with the ASSIGN or UNASSIGN parameters and the storage specified is below 16 megabytes, the address in the first fullword must begin on a page boundary. A page is 4096

bytes. The value of the second fullword indicates the number of pages below 16 megabytes that are to be assigned to a subspace.

Each storage range must reside in a single subpool.

Obtain your subspace storage by selecting a storage subpool with the storage attributes that subspaces require. The chapter on virtual storage in *z/OS MVS Programming: Authorized Assembler Services Guide* contains a table listing all subpools and the storage attributes associated with them. The following are the required and optional storage attributes for subspaces.

Table 38. Storage Attributes Required for Subspaces

Storage Attribute	Requirement	Comments
Location	Private	Subspace storage must be in high private or low private storage.
Fetch Protection	None	Subspace storage can be fetch-protected, but fetch-protection is not required.
Type	Pageable	Subspace storage must be pageable.
Owner	Task or job step	Subspace storage must be owned by the task creating the subspace, or a task higher in the task hierarchy.
Storage key	None	Subspace storage has no storage key requirements.

RANGLIST is a required parameter when you specify the IDENTIFY, UNIDENTIFY, ASSIGN, and UNASSIGN parameters.

,NUMRANGE=*numrange_addr*
Specifies the address of an optional fullword input variable that indicates the number of ranges in the range list. The number of ranges must be a least 1 and no more than 16. If you do not code NUMRANGE, the default number of ranges is 1, and the range list is limited to one entry.

NUMRANGE is an optional parameter when you specify the IDENTIFY, UNIDENTIFY, ASSIGN, or UNASSIGN parameters.

,NAME=*name_addr*
Specifies the address of the 8-byte variable or constant that contains the name of the subspace.

Subspace names are from 1 to 8 bytes long. They can contain letters, numbers, and @, #, and $, but they cannot contain embedded blanks. Names that are fewer than 8 bytes must be left-justified and padded on the right with blanks.

Unless you specify GENNAME=YES, the subspace name must begin with a letter or an @, #, or $ character. When you do not code GENNAME, or you specify either GENNAME=NO or GENNAME=COND, the name cannot begin with a number or be blank.

Subspace names must be unique within the home address space of the owning task. No two subspaces can have the same name. To ensure that the names for your subspaces are unique, code the GENNAME parameter to have the system generate a unique name. If you choose to let the system generate the subspace names for you, you must still supply three characters for the system to use.

NAME is a required parameter when you specify the CREATE parameter.

,GENNAME=NO
,GENNAME=COND
,GENNAME=YES
Specifies whether you want the system to generate a name for the subspace to ensure that all names are unique within the address space. The system generates a name by adding a 5-character prefix to the first three characters of the name you supply on the NAME parameter. For example, if you supply

'XYZDATA' on the NAME parameter, the name becomes '*ccccc*XYZ'. "ccccc" is the 5-character string generated by the system, and XYZ comes from the name you supplied on NAME.

The keywords that are valid for GENNAME and their meanings follow:

GENNAME=NO
The system does not generate a name. You *must* supply a name unique within the address space. GENNAME=NO is the default.

GENNAME=COND
The system generates a unique name only if you supply a name that is already being used. Otherwise, the system uses the name you supply.

GENNAME=YES
The system takes the name you supply on the NAME parameter and makes it unique. When you specify GENNAME=YES, the name you supply in the name parameter can begin with a numeric.

If you want the system to return the unique name it generates, use the OUTNAME parameter.

GENNAME is an optional parameter when you specify the CREATE parameter.

,OUTNAME=*outname_addr*
Specifies the address of the 8-byte variable into which the system returns the subspace name it generated, if you specify GENNAME=YES or GENNAME=COND. The OUTNAME parameter is optional when you specify the CREATE parameter.

,STOKEN=*stoken_addr*
Specifies the address of the 8-byte STOKEN for the subspace. The system returns an STOKEN value as output for a CREATE request. For other requests, you supply this returned value as input. STOKEN is a required parameter for the CREATE, ASSIGN, UNASSIGN, and DELETE requests.

ABEND codes

IARSUBSP might abnormally end with abend code X'3C6'. See *z/OS MVS System Codes* for an explanation and programmer response.

Return and reason codes

When the IARSUBSP macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes. GPR 0 contains one of the following hexadecimal reason codes.

Table 39. Return and Reason Codes for the IARSUBSP Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: The IARSUBSP request completed successfully. **Action**: None required.
04	xx0115xx	**Meaning**: IARSUBSP IDENTIFY completed successfully, but some ranges of storage had already been identified. **Action**: None required. However, you might want to take some action based on your application.
04	xx0315xx	**Meaning**: IARSUBSP ASSIGN completed successfully, but some of the storage specified on the RANGLIST parameter already had been assigned to the subspace indicated by the STOKEN parameter. **Action**: None required. However, you might want to take some action based on your application.

Table 39. Return and Reason Codes for the IARSUBSP Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
04	xx0415xx	**Meaning**: IARSUBSP UNASSIGN completed successfully, but one of the following conditions is true for some of the storage specified on the RANGLIST parameter: • Some storage already had been disassociated from the subspace by a previous UNASSIGN request • Some storage never had been assigned to a subspace. **Action**: None required. However, you might want to take some action based on your application.
04	xx0615xx	**Meaning**: IARSUBSP UNIDENTIFY completed successfully, but one of the following conditions is true for some of the storage specified on the RANGLIST parameter: • Some storage already had been made ineligible to be assigned to the subspace by a previous UNIDENTIFY request • Some storage never had been made eligible to be assigned to a subspace. **Action**: None required. However, you might want to take some action based on your application.
08	xx0212xx	**Meaning**: Environmental error. IARSUBSP CREATE failed. The system's set of generated names for subspaces has been temporarily exhausted. **Action**: Reissue IARSUBSP CREATE, specifying a unique name on the NAME parameter and GENNAME=NO. Or, issue IARSUBSP UNASSIGN and IARSUBSP DELETE for any subspaces that are no longer required, and reissue the CREATE request.
08	xx0213xx	**Meaning**: Program error. IARSUBSP CREATE failed. The name specified on the NAME parameter is not unique within the address space. **Action**: Change the name specified on the NAME parameter to a unique name, or specify GENNAME=COND or GENNAME=YES, and reissue the request.
08	xxFF00xx	**Meaning**: Environmental error. IARSUBSP failed. The system does not support subspaces. **Action**: Contact your system programmer to determine if the subspace group facility can be made available.
0C	xx0114xx	**Meaning**: Environmental error. IARSUBSP IDENTIFY failed. The system cannot perform any subspace services because of a shortage of resources. **Action**: Reissue the request. If the problem persists, contact your system programmer.
0C	xx0214xx	**Meaning**: Environmental error. IARSUBSP CREATE failed. The system cannot perform any subspace services because of a shortage of resources. **Action**: Reissue the request. If the problem persists, contact your system programmer.
0C	xx0314xx	**Meaning**: Environmental error. IARSUBSP ASSIGN failed. The system cannot perform any subspace services because of a shortage of resources. **Action**: Reissue the request. If the problem persists, contact your system programmer.
0C	xx0411xx	**Meaning**: System error. IARSUBSP UNASSIGN failed. One or more pages of storage were not processed. **Action**: Reissue the request. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel.

Example

For a complete example of creating, managing, and deleting subspaces, see the chapter on subspaces in *z/OS MVS Programming: Extended Addressability Guide*.

IARSUBSP - List form

Use the list form of the IARSUBSP macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

The list form of the IARSUBSP macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARSUBSP.
IARSUBSP	
ƀ	One or more blanks must follow IARSUBSP.
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default:** 0D

Parameters

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IARSUBSP macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IARSUBSP - Execute form

Use the execute form of the IARSUBSP macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the IARSUBSP macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.

Syntax	Description
ƀ	One or more blanks must precede IARSUBSP.
IARSUBSP	
ƀ	One or more blanks must follow IARSUBSP.
	Valid parameters (required parameters are underlined):
IDENTIFY	RANGLIST,NUMRANGE
CREATE	NAME,STOKEN,GENNAME,OUTNAME
ASSIGN	RANGLIST,STOKEN,NUMRANGE
UNASSIGN	RANGLIST,STOKEN,NUMRANGE
DELETE	STOKEN
UNIDENTIFY	RANGLIST,NUMRANGE
,RANGLIST=*ranglist_addr*	RS-type address, or address in register (2) - (12).
,NUMRANGE=*numrange_addr*	RS-type address, or address in register (2) - (12).
	Default: 1 range
,NAME=*name_addr*	RS-type address, or address in register (2) - (12).
,GENNAME=NO	**Default**: GENNAME=NO
,GENNAME=COND	
,GENNAME=YES	
,OUTNAME=*outname_addr*	RS-type address, or address in register (2) - (12).
,STOKEN=*stoken_addr*	RS-type address, or address in register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE
,MF=(E,*list addr*,NOCHECK)	

The parameters are explained under the standard form of the IARSUBSP macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)

Specifies the execute form of the IARSUBSP macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the macro is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

Chapter 33. IARVSERV — Request to share virtual storage

Description

Use the IARVSERV macro to define virtual storage areas to be shared by programs. This sharing can reduce the amount of processor storage required and the I/O necessary to support many applications that process large amounts of data. It also provides a way for programs executing in 24 bit addressing mode to access data residing above 16 megabytes.

Using IARVSERV allows programs to share data in virtual storage without the central storage constraints and processor overhead of other methods of sharing data. The type of storage access is controlled so that you can choose to allow read only or writing to the shared data with several variations. The type of storage access is called a **view**. Data to be shared is called the **source**. The source is the original data or the virtual storage that contains the data to be shared. This data is made accessible through an obtained storage area called the **target**. The source and target form a **sharing group**.

Through the IARVSERV macro, you can:

- Request that a virtual storage area (source) be eligible to be shared through a target view (SHARE parameter).
- Request that the source and targets no longer be shared (UNSHARE parameter).
- Request that the type of storage access to the data be changed.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about sharing data through the use of the IARVSERV macro. IARVSERV is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with PSW key that allows access to the source, target, or both, depending on the value specified through the TARGET_VIEW parameter. If the value specified on the NUMRANGE parameter is greater that 16, supervisor state or PSW key 0-7 is required. See *z/OS MVS Programming: Authorized Assembler Services Guide* for additional information.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31- or 64-bit.
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	The caller may hold the local lock, but is not required to hold any locks.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

- You must specify a range list that is mapped by the IARVRL macro. This is done using the RANGLIST parameter.
- If you specify more than 16 ranges, you must put the range list in fixed storage.
- The address space owing the source or targets must be swapped in when IARVSERV is issued if either the source or target area is:
  - in an address space other than the home address space of the caller, or
  - in a data space owned by an address space other than the home address of the caller.

  The address space must remain swapped in until the IARVSERV macro has completed.
- Before your program issues the IARVSERV macro, it must use the GETMAIN, STORAGE, or DSPSERV macro to obtain storage for the source, target, or both.
- Attributes for storage depend on the subpool specified on the GETMAIN, STORAGE, or DSPSERV macros. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information on virtual storage management and subpool attributes. The following table shows the permitted combinations of storage attributes supported for the source and target areas (with the exceptions as noted in Restrictions).

Table 40. IARVSERV Permitted Storage Combinations

Source Area	**Target Area**
Pageable	Pageable
Fixed in non-swappable storage with central storage below 16 megabytes	Any kind of storage
Fixed in non-swappable storage with central storage above 16 megabytes	Any storage that does not require the backing of central storage below 16 megabytes (if fixed)
Fixed in swappable storage	Any kind of storage, provided that TARGET_VIEW=UNIQUEWRITE parameter is specified

Restrictions

The following restrictions apply:

- For the SHARE parameter, the source area must not contain pages in the nucleus (read-only, extended read-only, read-write and extended read-write areas).
- For the SHARE parameter, the target area must not contain page-protected or page-fixed pages.
- For the UNSHARE parameter, the sharing group must not contain page protected-pages unless the RETAIN=YES parameteris specified. The sharing group must also not contain any page-fixed pages.
- The TPROT instruction cannot be used to determine whether the invoker has write access to views in a share group with unique-write views or to a target-write view. The TPROT instruction will indicate that these views are protected when the invoker has write access.

Input register information

Before issuing the IARVSERV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 contains a non-zero return code; otherwise, used as a work register by the system.

1
Used as a work register by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Return code.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

Take care when using the RETAIN=YES parameter value. With RETAIN=YES, storage is not returned to the system which reduces the amount available to the system and other programs, thus potentially affecting system performance.

In order to expedite the return of all internal control blocks for the shared storage back to the system, IBM recommends issuing IARVSERV UNSHARE against all views for both source and target that are originally shared. For an example of how to code the UNSHARE parameter, see *z/OS MVS Programming: Assembler Services Reference IAR-XCT*.

Syntax

The standard form of the IARVSERV macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARVSERV.
IARVSERV	
ƀ	One or more blanks must follow IARVSERV.

Syntax	Description
SHARE	
UNSHARE	
CHANGEACCESS	
,RANGLIST=*ranglist_addr*	*ranglist_addr*: RS-type address, or register (2) - (12).
,NUMRANGE=*numrange_addr*	*numrange_addr*: RS-type address, or register (2) - (12).
	Default: 1 range
,TARGET_VIEW=READONLY	
,TARGET_VIEW=SHAREDWRITE	
,TARGET_VIEW=UNIQUEWRITE	
,TARGET_VIEW=TARGETWRITE	
,TARGET_VIEW=LIKESOURCE	
,TARGET_VIEW=HIDDEN	
,COPYNOW	
,RETAIN=NO	**Default**: RETAIN=NO
,RETAIN=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0

Parameters

The SHARE, UNSHARE, and CHANGEACCESS parameters designate the services of the IARVSERV macro, and are mutually exclusive.

The parameters are explained as follows:

SHARE
Requests that the source be made shareable through the target to create a sharing group. When you issue the IARVSERV macro with SHARE, you must specify the RANGLIST and the TARGET_VIEW parameters. The NUMRANGE parameter is optional.

UNSHARE
Requests that the specified virtual storage no longer be used to access shared storage. When you issue the IARVSERV macro with UNSHARE, you must specify the RANGLIST parameter. The

NUMRANGE, and RETAIN parameters are optional. Using the RETAIN parameter can allow the target area data to remain available to other programs that can access the target area.

CHANGEACCESS
Requests that the type of access to the specified virtual storage be changed. When you issue the IARVSERV macro with CHANGEACCESS, you must specify the RANGLIST and the TARGET_VIEW parameters. The NUMRANGE parameter is optional.

,RANGLIST=*ranglist_addr*
Specifies the name (RS-type) or address (in register 2-12) of a required input fullword that contains the address of the range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 28 bytes long. A mapping of each entry is provided through the mapping macro IARVRL.

,NUMRANGE=*numrange_addr*
Specifies the name (RS-type) or address (in register 2-12) of an optional parameter that provides the number of entries in the supplied RANGLIST. Only authorized programs can specify more that 16 entries in the range list. If you do not specify NUMRANGE, the system assumes the range list contains only one entry.

,TARGET_VIEW=READONLY
,TARGET_VIEW=SHAREDWRITE
,TARGET_VIEW=UNIQUEWRITE
,TARGET_VIEW=TARGETWRITE
,TARGET_VIEW=LIKESOURCE
,TARGET_VIEW=HIDDEN
Specifies the way you want to share storage when used on storage not already part of a sharing group, or how you want to change or add storage access to the sharing group for storage already shared.

The keywords that are valid for TARGET_VIEW and their meanings follow:

READONLY
Specifies that the target can be used only to read shared data. Any attempt to alter shared data by writing into the target will cause a program check.

SHAREDWRITE
Specifies that the target can be used to read or modify shared data. When a program changes data in the target, the new data becomes visible among all those programs that have READONLY and SHAREDWRITE access to the source. Those programs with UNIQUEWRITE access to the source will not see the changed data.

UNIQUEWRITE
Specifies that the target can be used to read shared data and to retain a private copy of the shared data should the source or any target get altered. When another user of the target modifies the data, the page in the target containing the modified data becomes a private copy that is unique to that user (with UNIQUEWRITE) and not accessible to any other program.

TARGETWRITE
Specifies that the target can be used to read shared data and retain a private copy of the shared data if this view of the shared data is altered. When another user of the target area writes new data into the target area, any page in the target area containing the new data becomes a private copy that is unique and is not seen by to any other user. The page is no longer a member of any sharing group. The original source data is unchanged. When a SHAREDWRITE view of the data gets altered, the TARGETWRITE view will see those changes.

LIKESOURCE
Specifies that the view type for the new target area is to be the same as the current view of the source. If the source is not currently shared, a copy of the source is made to the new target as if COPYNOW had been coded.

HIDDEN
Specifies that the data in the target area will be inaccessible until the view type is changed to READONLY, SHAREDWRITE, UNIQUEWRITE, or TARGETWRITE. Any attempt to access a hidden target area will cause a program check.

,COPYNOW
Specifies whether the target should get a copy of the source data when using UNIQUEWRITE or LIKESOURCE. You can use COPYNOW only when you specify TARGET_VIEW=UNIQUEWRITE or TARGET_VIEW=LIKESOURCE.

,RETAIN=YES
,RETAIN=NO
Specifies whether a copy of the shared data is to be retained in the target after the system finishes processing the UNSHARE request.

RETAIN=YES
Specifies that the target view should retain a copy of the shared data. Using UNSHARE with RETAIN=YES requires the system to allocate new resources to back the target area.

RETAIN=NO
Specifies that the contents of the target area are unpredictable. To ensure zeroes, the user should issue a PGSER RELEASE or DSPSERV RELEASE on the area after unsharing it. RETAIN=NO is the default.

Note: PGRLSE, PGSER RELEASE, PGSER FREE with RELEASE=Y, and PGFREE RELEASE=Y may ignore some or all of the pages in the input range and will not notify the caller if this was done.

Any pages in the input range that match any of the following conditions will be skipped, and processing continues with the next page in the range:

- Storage is not allocated or all pages in a segment have not yet been referenced.
- Page is in PSA, SQA or LSQA.
- Page is V=R. Effectively, it's fixed.
- Page is in BLDL, (E)PLPA, or (E)MLPA.
- Page has a page fix in progress or a nonzero FIX count.
- Pages with COMMIT in progress or with DISASSOCIATE in progress.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

ABEND codes

IARVSERV might abnormally terminate with the abend code X'6C5'. See *z/OS MVS System Codes* for an explanation and programmer response.

Return and reason codes

When the IARVSERV macro returns control to your program, GPR 15 contains the return code. If the return code is not 0, GPR 0 contains the reason code.

Table 41. Return and Reason Codes for the IARVSERV Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: The IARVSERV request completed successfully. **Action**: None required.
04	xx0101xx	**Meaning** IARVSERV SHARE completed successfully. The processor does not support SHARE for UNIQUEWRITE. A unique copy of the target was made by the system. **Action**: None required.
04	xx0102xx	**Meaning**: IARVSERV SHARE completed successfully. However, the system found a condition that would lead to a storage requirement conflict for sharing with UNIQUEWRITE. For example, the source might be in non-pageable storage. A copy of the target was made by the system to avoid this conflict. **Action**: None required. However, you might want to correct the storage conflict.
04	xx0103xx	**Meaning**: IARVSERV SHARE found that some source pages were not obtained using the GETMAIN or STORAGE macros, or the source and target keys do not match and the request is for a UNIQUEWRITE target view. If the corresponding target pages were obtained using the GETMAIN or STORAGE macro, then they have been made first reference. **Action**: This is not necessarily an error. If you think you should not get this reason code, check program to be sure GETMAIN or STORAGE is issued and storage is of the same storage key for all source and target storage prior to using IARVSERV.
04	xx0203xx	**Meaning**: IARVSERV UNSHARE completed successfully. However, the system has overridden the RETAIN=NO option and kept a copy of the data in the target. **Action**: None required. However, you may want to correct your use of DIV.
04	xx0204xx	**Meaning**: IARVSERV UNSHARE completed successfully. The system has overridden the RETAIN=YES option because the shared data is associated with a DIV object, and the target area is not the original window mapped to the DIV object. The data in the target is unpredictable. **Action**: None required.

Table 41. Return and Reason Codes for the IARVSERV Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
04	xx0205xx	**Meaning**: IARVSERV UNSHARE completed successfully. Some pages in the target area no longer belong to any sharing group. This could be due to a copy being created by UNIQUEWRITE, or a second invocation of UNSHARE on the same view. **Action**: None required.
04	xx0301xx	**Meaning**: IARVSERV CHANGEACCESS completed successfully. The processor does not support CHANGEACCESS for UNIQUEWRITE, and a unique copy of the target page was made. **Action**: None required.
04	xx030Cxx	**Meaning**: IARVSERV CHANGEACCESS completed successfully. The system processed a CHANGEACCESS request for UNIQUEWRITE or TARGETWRITE for non-shared pages as a SHAREDWRITE request. **Action**: None required.
08	xx0104xx	**Meaning**: Environmental error. An unauthorized user attempted to share more pages than allowed by the installation (as defined through the installation exit IEFUSI). **Action**: Contact your system programmer to find out your installation limit and reduce the number of shared pages.
08	xx0105xx	**Meaning**: Environmental error. IARVSERV SHARE was requested with TARGETWRITE, but the SOP hardware feature was not available. **Action**: Contact your system programmer to find out when the SOP feature might become available.
08	xx0305xx	**Meaning**: Environmental error. IARVSERV CHANGEACCESS was requested with TARGETWRITE, but the SOP hardware feature was not available. **Action**: Contact your system programmer to find out when the SOP feature may become available.
0C	xx010Axx	**Meaning**: Environmental error. IARVSERV SHARE cannot complete the request because of a shortage of resources. **Action**: Retry the request one or more times to see if resources become available. Contact the system programmer to determine resources available to you.
0C	xx013Cxx	**Meaning**: System error. IARVSERV SHARE cannot complete the request because a required page is unavailable or lost. **Action**: Check the paging data set for possible I/O errors. Refer to X'028' abend description in *z/OS MVS System Codes* for paging error advice.

Table 41. Return and Reason Codes for the IARVSERV Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
0C	xx020Bxx	**Meaning**: System error. IARVSERV UNSHARE cannot complete the request because of a required page being unavailable or lost. **Action**: Check the logrec data set for possible I/O errors. Refer to X'028' abend description in *z/OS MVS System Codes* for paging error advice.
0C	xx030Bxx	**Meaning**: System error. IARVSERV CHANGEACCESS cannot complete the request because of a required page being unavailable or lost. **Action**: Check the logrec data set for possible I/O errors. Refer to X'028' abend description in *z/OS MVS System Codes* for paging error advice.

Example 1

Issue a request to share eight pages as read-only, and use a register to specify the address of the range list.

```
SERV1    IARVSERV SHARE,RANGLIST=(4),TARGET_VIEW=READONLY
*
         IARVRL
```

Example 2

Issue UNSHARE for the pages in Example 1, and specify that the system is not to retain the shared data.

```
SERV2    IARVSERV UNSHARE,RANGLIST=(4),RETAIN=NO
*
         IARVRL
```

Example 3

Issue a request to share pages as read-only, and use an RS-type address to specify the location of the range list address.

```
SERV3    IARVSERV SHARE,RANGLIST=VRLPTR,TARGET_VIEW=READONLY
*
VRLPTR   DC    A(MYVRL1)
MYVRL1   DS    7F
         IARVRL
```

Example 4

Issue a request to share pages as target write.

```
SERV4    IARVSERV SHARE,RANGLIST=(5),TARGET_VIEW=TARGETWRITE
*
         IARVRL
```

Example 5

Issue a request to change access for hidden.

```
SERV5    IARVSERV CHANGEACCESS,RANGLIST=(5),TARGET_VIEW=HIDDEN
*
         IARVRL
```

IARVSERV—List form

Use the list form of the IARVSERV macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

The list form of the IARVSERV macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARVSERV.
IARVSERV	
ƀ	One or more blanks must follow IARVSERV.
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default**: 0D

The parameters are explained under the standard form of the IARVSERV macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IARVSERV macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IARVSERV - Execute form

Use the execute form of the IARVSERV macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the IARVSERV macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARVSERV.
IARVSERV	
ƀ	One or more blanks must follow IARVSERV.
SHARE	
UNSHARE	
CHANGEACCESS	
,RANGLIST=*ranglist_addr*	*ranglist_addr*: RS-type address, or address in register (2) - (12).
,NUMRANGE=*numrange_addr*	*numrange_addr*: RS-type address, or address in register (2) - (12).
	Default: 1 range
,TARGET_VIEW=READONLY	
,TARGET_VIEW=SHAREDWRITE	
,TARGET_VIEW=UNIQUEWRITE	
,TARGET_VIEW=TARGETWRITE	
,TARGET_VIEW=LIKESOURCE	
,TARGET_VIEW=HIDDEN	
,COPYNOW	
,RETAIN=NO	**Default**: RETAIN=NO
,RETAIN=YES	
,PLISTVER=IMPLIED_VERSION	

Syntax	Description
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

The parameters are explained under the standard form of the IARVSERV macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
Specifies the execute form of the IARVSERV macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

Chapter 34. IARV64 — 64–bit virtual storage allocation

Description

The IARV64 macro allows a program to use the full range of virtual storage in an address space that is supported by 64-bit addresses. The macro creates and frees storage areas above the two-gigabyte address and manages the physical frames behind the storage. Each storage area is a multiple of one megabyte in size and begins on a megabyte boundary. You can think of the IARV64 macro as the GETMAIN, FREEMAIN, PGSER, and STORAGE macro for virtual storage above the two-gigabyte address.

The two-gigabyte address in the address space is marked by a virtual line called the *bar*. The bar separates storage below the two-gigabyte address, called *below the bar*, from storage above the two-gigabyte address, called *above the bar*. Programs use the IARV64 macro to obtain storage above the bar in "chunks" of virtual storage called *memory objects*. Your installation can set a limit on the use of the address space above the bar for a single address space. The limit is called the MEMLIMIT.

When you create a nonshared, non-2 GB memory object, you can specify a guard area (not accessible) and a usable area. Subsequently, you can create alternate guard areas or change all or some of a guard area into an accessible area, or vice versa.

The following services are provided:

GETSTOR
Create a private memory object, in "REQUEST=GETSTOR option of IARV64" on page 370.

GETSHARED
Create a memory object that can be shared across multiple address spaces, in "REQUEST=GETSHARED option of IARV64" on page 387.

GETCOMMON
Create a 64-bit common memory object, in "REQUEST=GETCOMMON option of IARV64" on page 395.

CHANGEACCESS
Request that a view type for segments within the specified shared memory objects be changed, in "REQUEST=CHANGEACCESS option of IARV64" on page 408.

CHANGEATTRIBUTE
Request to change an attribute of storage within one or more memory objects, in REQUEST=CHANGEATTRIBUTE option of IARV64.

CHANGEGUARD
Request that a specified range in a nonshared, non-2GB memory object be changed from guard area to usable area or vice versa, in "REQUEST=CHANGEGUARD option of IARV64" on page 418.

COUNTPAGES
Count the number of 4K pages currently in use in real storage, on auxiliary storage, and in both real storage and on auxiliary storage to back the input high virtual storage ranges, in "REQUEST=COUNTPAGES option of IARV64" on page 425.

DETACH
Free one or more memory objects. For a nonshared memory object, the object is freed. For a shared memory object, the object is freed only when the last shared user of that memory object issues the DETACH (this includes a DETACH corresponding to the system attachment formed when the object was created through GETSHARED). See "REQUEST=DETACH option of IARV64" on page 431.

DISCARDDATA
Discard data within physical pages of one or more memory objects. in "REQUEST=DISCARDDATA option of IARV64" on page 440.

LIST
Request a list of memory objects, in "REQUEST=LIST option of IARV64" on page 446.

PAGEFIX
Fix physical pages within one or more nonshared memory objects, in "REQUEST=PAGEFIX option of IARV64" on page 456.

PAGEIN
Notify the system that data within physical pages of one or more memory objects are needed in the near future, in "REQUEST=PAGEIN option of IARV64" on page 462.

PAGEOUT
Notify the system that data within physical pages of one or more memory objects will not be used in the near future, in "REQUEST=PAGEOUT option of IARV64" on page 467.

PAGEUNFIX
Unfix physical pages within one or more nonshared memory objects, in "REQUEST=PAGEUNFIX option of IARV64" on page 472.

PROTECT
Request that data within one or more memory objects be made read-only, in "REQUEST=PROTECT option of IARV64" on page 478.

SHAREMEMOBJ
Request that the specified address space be given access to one or more specified shared memory objects, in"REQUEST=SHAREMEMOBJ option of IARV64" on page 484

UNPROTECT
Request that data within one or more memory objects be made modifiable, in "REQUEST=UNPROTECT option of IARV64" on page 490.

For guidance information about the use of 64–bit virtual storage allocation, see *z/OS MVS Programming: Extended Addressability Guide*.

After the separate descriptions of each individual Request are the following topics which apply to all of the Requests:

- The abend codes in "ABEND codes" on page 496,
- The return and reason codes in "Return and reason codes" on page 496, and
- Examples of using IARV64 in "Example" on page 498

Note: The examples apply to REQUEST=GETSTOR, PAGEFIX, PAGEUNFIX, and DETACH.

Facts associated with these services:

- A segment represents one megabyte of virtual storage starting on a megabyte boundary.
- The storage returned by the GETSTOR, GETSHARED, or GETCOMMON services is called a *memory object*.
- The storage returned by GETSHARED is referred to as a *shared memory object*.
- The storage returned by GETSTOR is referred to as a *private memory object* or a *system memory object*.
- The storage returned by GETCOMMON is referred to as a *common memory object*.
- The limit of storage per address space allowed to be used above the bar is called the MEMLIMIT. This is similar to the REGION parameter for storage below the bar. The following categories of storage do not count against the MEMLIMIT:
  - The guard area in a memory object
  - The storage created by IARV64 GETSTOR with LOCALSYSAREA=YES
  - Shared memory objects, such as storage created by IARV64 GETSHARED

REQUEST=GETSTOR option of IARV64

REQUEST=GETSTOR allows you to create a memory object.

Note: For more information on GETSTOR requests, see the *z/OS MVS Programming: Extended Addressability Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15. To use the PAGEFRAMESIZE parameter, a caller can be in problem state with either one of the following authorizations: APF-authorized Authorized for read to IARRSM.LRGPAGES
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN **Note:** The problem state caller running in PSW key 8-15 can use GETSTOR/DETACH only when the primary address space is the home address space.
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	• Enabled for I/O and external interrupts. • Disabled for 64-bit common memory objects allocated with TYPE=DREF. • Disabled for TYPE=PAGEABLE and the storage is in the first reference state.
Locks:	A local lock may be held, subject to the following limitation: When a local lock is held for a request (GETSTOR, SHAREMEMOBJ, DETACH, CHANGEGUARD, or DISCARDDATA) the lock must be for the address space specified or set as the default by the input ALETVALUE.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=GETSTOR option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=GETSTOR	
,COND=NO	**Default:** COND=NO
,COND=YES	

Syntax	Description
,LOCALSYSAREA=NO	**Default:** LOCALSYSAREA=NO
,LOCALSYSAREA=YES	
,SEGMENTS=*'segments'*	*segments*: RS-type address or address in register (2) - (12).
,TYPE=DREF	
,TYPE=PAGEABLE	**Default:** TYPE=PAGEABLE when one of the following is specified: • PAGEFRAMESIZE=PAGEABLE1MEG • PAGEFRAMESIZE=4K • PAGEFRAMESIZE=MAX and the memory object is backed with 4 KB-page frames. When PAGEFRAMESIZE=DREF1MEG is specified, the default value is TYPE=DREF. If PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX is specified and the memory object is backed with 1 MB-page frames, the 1 MB-pages backing this memory object are fixed.
,PAGEFRAMESIZE=4K	**Default:** PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1MEG	
,PAGEFRAMESIZE=MAX	
,PAGEFRAMESIZE=PAGEABLE1MEG	
,PAGEFRAMESIZE=DREF1MEG	
,UNITS=*units*	*units:* Size of the memory object, which is the number of units specified by UNITSIZE.
,UNITSIZE=1M	Specifies a 1 MB unit size.
,PAGEFRAMESIZE=4K|1M	If UNITSIZE=1M is specified, a PAGEFRAMESIZE of 4K or 1M must be specified. There is no default value.
,UNITSIZE=2G	Specifies a 2 GB unit size.
,PAGEFRAMESIZE=4K|1M|2G	If UNITSIZE=2G is specified, a PAGEFRAMESIZE of 4K, 1M or 2G must be specified. There is no default value.
,TYPE=FIXED	
,TYPE=DREF	
,TYPE=PAGEABLE	
,KEY=*key*	*key:* RS-type address or address in register (2) - (12).

Syntax	Description
,KEY=CALLERKEY	**Default:** KEY=CALLERKEY
,FPROT=YES	**Default:** FPROT=YES
,FPROT=NO	
,SENSITIVE=UNKNOWN	**Default:** SENSITIVE=UNKNOWN
,SENSITIVE=YES	
,SENSITIVE=NO	
,MEMLIMIT=YES	**Default:** MEMLIMIT=YES
,MEMLIMIT=NO	
,MEMLIMIT=COND	
,SVCDUMPRGN=YES	**Default:** SVCDUMPRGN=YES
,SVCDUMPRGN=NO	
,DUMP=LIKERGN	**Default:** DUMP=LIKERGN
,DUMPPRIORITY=99	**Default:** DUMPPRIORITY=99
,DUMPPRIORITY=*dumppriority*	
,DUMP=LIKELSQA	
,DUMP=NO	
,DUMP=BYOPTIONVALUE	
,OPTIONVALUE=*option*	*option:* RS-type address or address in register (2) - (12).
,CONTROL=UNAUTH	**Default:** CONTROL=UNAUTH
,CONTROL=AUTH	
,MOTKNSOURCE=USER	**Default:** MOTKNSOURCE=USER
,MOTKN=motkn	**Default:** MOTKN
,MOTKNCREATOR=USER	**Default:** MOTKNCREATOR=USER
,MOTKNCREATOR=SYSTEM	

Syntax	Description
,USERTKN=NO_USERTKN	**Default:** USERTKN=NO_USERTKN
,USERTKN=*usertkn*	*usertkn:* RS-type address or address in register (2) - (12).
,MOTKNSOURCE=SYSTEM	
,OUTMOTKN=*outmotkn*	*Outmotkn:* RS-type address or address in register (2) - (12).
,USERTKN=*usertkn*	*usertkn:* RS-type address or address in register (2) - (12).
,USERTKN=NO_USERTKN	**Default:** USERTKN=NO_USERTKN
,GUARDSIZE=*guardsize*	*guardsize:* RS-type address or address in register (2) - (12).
,GUARDSIZE=0	**Default:** GUARDSIZE=0
,GUARDSIZE64=*guardsize64*	*guardsize64:* RS-type address or address in register (2) - (12).
,GUARDSIZE64=0	**Default:** GUARDSIZE64=0
,GUARDLOC=LOW	**Default:** GUARDLOC=LOW
,GUARDLOC=HIGH	
,TTOKEN=*ttoken*	*ttoken:* RS-type address or address in register (2) - (12).
,TTOKEN=NO_TTOKEN	**Default:** TTOKEN=NO_TTOKEN
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,ORIGIN=*origin*	*origin:* RS-type address or address in register (2) - (12).
,DETACHFIXED=NO	**Default:** DETACHFIXED=NO
,DETACHFIXED=YES	
,SADMP=DEFAULT	**Default:** SADMP=DEFAULT
,SADMP=YES	
,SADMP=NO	

Syntax	Description
,INORIGIN=*inorigin*	*inorigin:* RS-type address or address in register (2) - (12).
,INORIGIN=NO_INORIGIN	**Default:** INORIGIN=NO_INORIGIN
,USE2GTO32G=NO	**Default:** USE2GTO32G=NO
,USE2GTO32G=YES	
,USE2GTO64G=NO	**Default:** USE2GTO64G=NO
,USE2GTO64G=YES	
,EXECUTABLE=SYSTEM_RULES	**Default:** EXECUTABLE=SYSTEM_RULES
,EXECUTABLE=YES	
,EXECUTABLE=NO	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4,5	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=GETSTOR
A required parameter. REQUEST=GETSTOR creates a private memory object if LOCALSYSAREA=YES is not specified. If LOCALSYSAREA=YES is specified, then a system memory object is returned. The storage obtained in the system area using the LOCALSYSAREA keyword will not be copied during Fork

processing. The use of local system area storage does not preclude checkpoint from succeeding. At completion, the memory object is created in the address space you indicate.

,COND=NO
,COND=YES
An optional input parameter that specifies whether the request is unconditional or conditional. The default is COND=NO.

,COND=NO
The request is unconditional. The request is abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request is not abnormally ended for resource unavailability.

When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete but a return code will be set to indicate that the request could not be completed successfully. Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work.

,LOCALSYSAREA=NO
,LOCALSYSAREA=YES
An optional input parameter that specifies whether this is an explicit allocation request for 64-bit virtual storage in local system area. The localsysarea parameter can be used only by callers running in supervisor state or with a PSW key 0-7. The default is LOCALSYSAREA= NO.

,LOCALSYSAREA=NO
The request will not be satisfied from the local system area.

,LOCALSYSAREA=YES
The request is to be satisfied from the local system area. The storage obtained with this keyword will not be copied during Fork processing. The use of local system area storage does not preclude checkpoint from succeeding.

,KEY=*key*
,KEY=CALLERKEY
An optional input parameter that specifies the storage key to be assigned to the memory object. The key must be in bits 0-3 of the specified byte. Bits 4-7 are ignored. The KEY parameter can be used only by callers running in supervisor state or with a PSW key 0-7; with the following exception: a PSW key 8 caller can specify a storage key of the memory object to be key 9.

If the key is not specified, the storage key of the memory object is the same as the caller's PSW key. The default is CALLERKEY.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,FPROT=YES
,FPROT=NO
An optional input parameter that specifies whether the memory object should be fetch-protected. The default is FPROT=YES.

,FPROT=YES
The entire memory object is fetch-protected. A program must have a PSW key that matches the storage key of the memory object (or have PSW key 0) to reference data in the memory object.

,FPROT=NO
The memory object is not fetch-protected.

,SENSITIVE=UNKNOWN
,SENSITIVE=YES
,SENSITIVE=NO
An optional keyword input that specifies whether the memory object will contain data that is potentially sensitive (for instance, private or confidential). The default is SENSITIVE=UNKNOWN.

,SENSITIVE=UNKNOWN
Specifies that the sensitivity of the data is not known. The memory object may or may not contain sensitive data.

,SENSITIVE=YES
Specifies that the memory object contains sensitive data.

,SENSITIVE=NO
Specifies that the memory object does not contain sensitive data.

,MEMLIMIT=YES
,MEMLIMIT=NO
,MEMLIMIT=COND
An optional input parameter that specified whether the allocation of the 64-bit memory object is to count towards the address space MEMLIMIT. The default is MEMLIMIT=YES.

,MEMLIMIT=YES
The 64-bit private memory object contributes towards the address space MEMLIMIT.

,MEMLIMIT=NO
The 64-bit private memory object is not counted against the address space MEMLIMIT. MEMLIMIT=NO is effective only when specified by authorized callers in supervisor state or key 0-7. Requests for MEMLIMIT=NO by unauthorized callers will result in a DC2 abend with reason code 19.

,MEMLIMIT=COND
If the caller is running in supervisor state or key 0 - 7, treat as MEMLIMIT=NO; otherwise, treat as MEMLIMIT=YES.

,SEGMENTS=*segments*

SEGMENTS and UNITS are mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the size of the memory object requested, in megabytes. This must be a nonzero value. The amount of storage requested that is not in the guard state is charged against the MEMLIMIT for the address space where the memory object is to be created.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1MEG
,PAGEFRAMESIZE=MAX
,PAGEFRAMESIZE=PAGEABLE1MEG
,PAGEFRAMESIZE=DREF1MEG
An optional input parameter that specifies the size of the page frames to back the virtual storage mapped by the allocated memory object.

,PAGEFRAMESIZE=4K
The memory object is backed by 4 KB-page frames, if available. This is the default value.

,PAGEFRAMESIZE=1MEG
The memory object is backed by 1 MB-page frames, if available.

,PAGEFRAMESIZE=MAX
The memory object is backed by the largest page frame size that is supported and available. Otherwise, the object is backed by 4 KB-page frames. 1 MB-page frames are backed at allocation and cannot be paged out to AUX. 4 KB-page frames are backed at first reference and can be paged out to AUX as long as TYPE=DREF is not specified.

,PAGEFRAMESIZE=PAGEABLE1MEG
The memory object is backed by pageable 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.

,PAGEFRAMESIZE=DREF1MEG
The memory object is backed by Dref 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.

,TYPE=PAGEABLE
,TYPE=DREF

An optional input parameter that specifies the type of storage that is requested. The default value is TYPE=PAGEABLE when one of the following parameters is specified:

- PAGEFRAMESIZE=4K
- PAGEFRAMESIZE=PAGEABLE1MEG
- PAGEFRAMESIZE=MAX and the memory object is backed with 4 KB-page frames.

The default value is TYPE=DREF when PAGEFRAMESIZE=DREF1MEG is specified. If PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX is specified and the memory object is backed with 1 MB-page frames, the 1 MB-pages backing this memory object are fixed.

Note:

1. When the memory object is backed by 4 KB-page frames, the 4 KB-pages backing this memory object are pageable if TYPE=DREF is not specified, or are fixed if TYPE=DREF is specified. The 4 KB-pages are backed at first reference and can only be paged out to AUX if TYPE=DREF is not specified.
2. When the memory object is backed by 1 MB-page frames as a result of PAGEFRAMESIZE=PAGEABLE1MEG or PAGEFRAMESIZE=DREF1MEG being specified, the 1 MB-pages backing this memory object are pageable if PAGEABLE1MEG is specified or fixed if DREF1MEG is specified. Pageable 1 MB-pages are backed at first reference and can be paged out to AUX. DREF 1 MB-pages are backed at first reference and are fixed—they cannot be paged out to AUX.
3. When the memory object is backed by 1 MB-page frames because PAGEFRAMESIZE=1MEG or PAGEFRAMSIZE=MAX has been specified, the 1 MB-pages backing this memory object are fixed. Pages are backed at allocation time and cannot be paged out to AUX.

,UNITS=*units*

UNITS and SEGMENTS are mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the size of the memory object as a number of units specified by the UNITSIZE parameter. This must be a nonzero value. The amount of storage requested that is not in the guard state is counted towards the MEMLIMIT for the address space where the memory object will be created. UNITS belongs to a set of mutually exclusive keys. This set is required; only one key can be specified.

,TYPE=PAGEABLE
,TYPE=DREF
,TYPE=FIXED

A required input parameter that specifies the type of requested storage.

,TYPE=PAGEABLE
Pages backing this memory object are pageable. Pages are backed at first reference and can be paged out to auxiliary storage. Virtual address ranges within the memory object can be explicitly fixed after allocation by using the IARV64 REQUEST=PAGEFIX request. TYPE=PAGEABLE is not valid with PAGEFRAMESIZE=2G.

,TYPE=DREF
Pages are backed in real memory at first reference, unless DREF storage is not available, in which case the program is ABENDed. Once backed, pages belonging to memory objects of TYPE=DREF remain in real storage and are never paged out to auxiliary storage. The memory object can be referenced while running disabled. The DREF attribute applies to the entire memory object. TYPE=DREF is not valid with PAGEFRAMESIZE=2G.

,TYPE=FIXED
Pages are backed in real storage immediately, unless fixed storage is not immediately available, in which case the request fails. Pages belonging to memory objects of TYPE=FIXED remain in real storage and are never be paged out to auxiliary storage. The memory object can

be referenced while running disabled. The FIXED attribute applies to the entire memory object when it is allocated. TYPE=FIXED is not valid with PAGEFRAMESIZE=4K.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,UNITSIZE=1M
,UNITSIZE=2G

A required input parameter that specifies the size for the UNITS parameter: either 1M or 2 GB.

,UNITSIZE=1M

Specifies that the memory object is in one-megabyte (1 MB) units. For example, a request for UNITS=3 with UNITSIZE=1M is a request for three megabytes of virtual storage starting on a 1 MB boundary. When UNITSIZE=1M is specified, one of the following PAGEFRAMESIZE values must also be specified:

PAGEFRAMESIZE=4K
PAGEFRAMESIZE=1M

A required input parameter that specifies the size of the page frames used to back the virtual storage mapped by the allocated memory object.

PAGEFRAMESIZE=4K

Specifies to back the memory object with 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported.

PAGEFRAMESIZE=1M

Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available, the system attempts to back the memory object at a smaller page frame size of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

,UNITSIZE=2G

Specifies that the memory object is in two-gigabyte (2G) units. For example, a request for UNITS=3 with UNITSIZE=2G is a request for six gigabytes of virtual storage starting on a 2 GB boundary.

PAGEFRAMESIZE=4K|1M|2G

A required input parameter that specifies the size of the page frames that back the virtual storage mapped by the allocated memory object.

PAGEFRAMESIZE=4K

Specifies to back the memory object by 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported with this value.

PAGEFRAMESIZE=1M

Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available when TYPE=PAGEABLE or TYPE=DREF is requested, the system attempts to back the memory object using a smaller page frame size of the specified TYPE. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

PAGEFRAMESIZE=2G

Specifies to back the memory object by two-gigabyte (2 GB) FIXED page frames. PAGEFRAMESIZE=2G is valid only when TYPE=FIXED is specified. If 2 GB page frames are not supported or not available, the request fails.

,SVCDUMPRGN=YES
,SVCDUMPRGN=NO

SVCDUMPRGN and DUMP are mutually exclusive keys. This set is optional; only one key may be specified.

An optional input parameter that specifies whether the memory object should be included in an SVC dump when region is requested. The default is SVCDUMPRGN=YES.for TYPE=PAGEABLE. If neither

the SVCDUMPRGN keyword nor the DUMP keyword is specified the defaults that apply are as described under the defaults for the DUMP keyword.

,SVCDUMPRGN=YES
The memory object is included in an SVC dump when RGN is dumped. This is equivalent to DUMP=LIKERGN.

,SVCDUMPRGN=NO
The memory object is only included in an SVC dump when explicitly requested.

,DUMP=LIKERGN
,DUMP=LIKELSQA
,DUMP=NO
,DUMP=BYOPTIONVALUE

DUMP and **SVCDUMPRGN** are mutually exclusive keys. This set is optional; only one key may be specified.

An optional input parameter that specifies whether the 64-bit private memory object will be included in an SVC dump when RGN or LSQA is dumped. When TYPE=PAGEABLE is specified on IARV64 GETSTOR the default is DUMP=LIKERGN. When TYPE=DREF is specified on IARV64 GETSTOR the default is DUMP=LIKELSQA. For memory objects backed with large pages the default is DUMP=NO.

,DUMP=LIKERGN
The 64-bit private memory object is included in an SVC dump when RGN is dumped.

,DUMPPRIORITY=99
,DUMPPRIORITY=*dumppriority*
An optional input parameter that specifies the dump priority of the memory object. This must be a non-zero value in the range of 1 to 99, with 1 being the highest priority and 99 being the lowest. The default is DUMPPRIORITY=99.

,DUMPPRIORITY=99
The dump priority of the memory object is 99 which is the lowest priority.

,DUMPPRIORITY=*dumppriority*
This parameter is the name (RS-type), or address in register (2)-(12), of an optional byte input that specifies the dump priority of the memory object. This must be a non-zero value in the range of 1 to 99, with 1 being the highest priority and 99 being the lowest.

,DUMP=LIKELSQA
The 64-bit private memory object is included in an SVC dump when RGN is dumped.

,DUMP=NO
The 64-bit private memory object is only included in an SVC dump when explicitly requested.

,DUMP=BYOPTIONVALUE
The 64-bit private memory object is dumped according to the option specified by the OPTIONVALUE keyword.

,OPTIONVALUE=*optionvalue*
This parameter is the name of a required one-byte integer input that contains one of the dump option values as specified by the bit constants.

,CONTROL=UNAUTH
,CONTROL=AUTH
An optional input parameter that specifies when the memory object should be eligible for the certain other services.

This is a permanent attribute of the memory object and cannot be altered by other services. The default is CONTROL=UNAUTH.

CONTROL=UNAUTH
The memory object can be freed by an unauthorized caller that owns the memory object. The memory object is NOT eligible for PAGEFIX.

CONTROL=AUTH
The memory object can be freed only by an authorized caller. The memory object is eligible for PAGEFIX and PAGEUNFIX (note that PAGEFIX and PAGEUNFIX still require an authorized caller). AUTH can be used only by callers running in supervisor state or with PSW key 0-7.

MOTKNSOURCE=USER
MOTKNSOURCE=SYSTEM
An optional input parameter that indicates who provided (or will provide) the memory object token.

MOTKNSOURCE=USER

The user provides the memory object token.

The following is a set of mutually exclusive keys. This set is optional; only one key may be specified.

MOTKN=*motkn*

This parameter belongs to a set of mutually exclusive keys. It is the name of an optional doubleword integer input that identifies the user token to be associated with the memory object. This can be used on a later DETACH request to free all memory objects associated with this value.

- To request a system-generated token, use:

```
IARV64 REQUEST(GETCOMMON) MOTKNSOURCE(SYSTEM) OUTMOTKN(mytoken)
```

- Use the returned token on subsequent IARV64 GETCOMMON requests, in order to associate other memory objects with the same token:

```
IARV64 REQUEST(GETCOMMON) MOTKNSOURCE(USER) MOTKN(mytoken)
```

To avoid inadvertent collisions in the values specified, the left word (bits 0-31) of the user token must be binary zeros for a problem state program. The system enforces this requirement. The right word (bits 32-63) should represent the virtual address of some storage related to the caller, which could be a control block address, an entry point address, and so on, which is used as an application choice.

The convention for supervisor state program is that the left word (bits 0-31) should represent an address of some storage related to the caller. The system enforces the rule that the left word is nonzero for supervisor state callers. The format for the right word (bits 32-63) is a choice left to the caller.

If you specify no user token, the default is that no user token is supplied to associate this memory object with others.

MOTKNCREATOR=USER
MOTKNCREATOR=SYSTEM
This parameter is an optional input parameter that indicates who created the memory object token

MOTKNCREATOR=USER
The memory object token is user-created.

MOTKNCREATOR=SYSTEM
The memory object token is system-created.

USERTKN=usertkn
USERTKN=NO_USERTKN
This parameter belongs to a set of mutually exclusive keys. It is the name of an optional doubleword integer input that is a synonym for MOTKN. You can use either USERTKN or MOTKN interchangeably.

MOTKNSOURCE=SYSTEM
The system provides the memory object token.

OUTMOTKN=*xoutmotkn*
This parameter is the name of a required doubleword integer output that identifies the user token to be associated with the memory object to be created by the system.

,USERTKN=*usertkn*
,USERTKN=NO_USERTKN
An optional input parameter that identifies the user token to be associated with the memory object. This can be used on a later DETACH request to free all memory objects associated with this value.

To avoid inadvertent collisions in the values specified, the left word (bits 0-31) of the user token must be binary zeros for a problem state program. The system enforces this requirement. The right word (bits 32-63) should represent the virtual address of some storage related to the caller, which could be a control block address, an entry point address, and so on, which is used as an application choice.

The convention for supervisor state program is that the left word (bits 0-31) should represent an address of some storage related to the caller. The system enforces the rule that the left word is nonzero for supervisor state callers. The format for the right word (bits 32-63) is a choice left to the caller.

If you specify NO_USERTKN, the default is that no user token is supplied to associate this memory object with others. The default is NO_USERTKN.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,GUARDSIZE=*guardsize*
,GUARDSIZE=0

GUARDSIZE and GUARDSIZE64 are mutually exclusive keys. This set is optional; only one key may be specified. A fullword integer input parameter that indicates the number of megabytes of guard area to be created at the high or low end of the memory object. Guard areas cannot be referenced and when referenced will cause a program check. Guard area does not count against the MEMLIMIT. A guard area can be reduced through CHANGEGUARD CONVERT=FROMGUARD.

GUARDSIZE must not be larger than the size of the memory object. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,GUARDSIZE64=*guardsize64*
,GUARDSIZE64=0

GUARDSIZE64 belongs to a set of mutually exclusive keys. This set is optional; only one key may be specified. A doubleword integer input parameter that indicates the number of megabytes of guard area to be created at the high or low end of the memory object. Guard areas cannot be referenced and when referenced will cause a program check. Guard area does not count against the MEMLIMIT. A guard area can be reduced through CHANGEGUARD CONVERT=FROMGUARD.

GUARDSIZE64 must not be larger than the size of the memory object. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,GUARDLOC=LOW
,GUARDLOC=HIGH
An optional input parameter that specifies whether the guard location is at the low virtual end of the memory object or the high virtual end. The default is GUARDLOC=LOW.

GUARDLOC=LOW
The guard areas are created starting from the origin of the memory object, that is, from the low virtual end.

GUARDLOC=HIGH
The guard areas are created at the end of the memory object, that is, at the high virtual end.

,TTOKEN=*ttoken*
,TTOKEN=NO_TTOKEN
An optional input parameter that identifies the task to assume ownership of the memory object. The TTOKEN is returned by the TCBTOKEN macro.

If TTOKEN is specified, the task identified by the TTOKEN becomes the owner of the memory object. If TTOKEN is not specified, the currently dispatched task becomes the owner of the memory object. The task identified by the TTOKEN must be in the address space specified or defaulted by the ALETVALUE keyword.

The TTOKEN parameter must be used by an caller that is an SRB.

When the TTOKEN parameter is used by problem state program with PSW key 8 - 15, the target task must represent the calling task OR the jobstep task for the calling task OR the mother task. A caller cannot assign ownership to a task above the jobstep task.

A memory object will be freed when its owning task terminates.

If the TTOKEN parameter is not specified, and the caller is a task (rather than an SRB), the currently dispatched task will become the owner of the memory object. An SRB will be abnormally ended if the TTOKEN parameter does not specify a valid TTOKEN. The default is NO_TTOKEN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16 character field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the memory object is to be created.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ORIGIN=*origin*
A required output parameter that contains the lowest address of the memory object. Note that when GUARDLOC=LOW is specified, the lowest address will point to a guard area which will cause an ABEND if referenced. For GUARDLOC=LOW the first usable area is the origin plus the size of the guard area.

Note: If MEMLIMIT is exceeded and RETCODE=8, then the address will be set to 7FFFF000 to cause an ABEND if referenced.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

DETACHFIXED=NO
DETACHFIXED=YES
An optional input parameter that specifies whether the memory object can be detached when it contains fixed pages at the time of the DETACH request. The default value for DETACHFIXED is NO.

DETACHFIXED=NO
The memory object will not be detached if it has any fixed pages when it is being detached.

DETACHFIXED=YES
The memory object will be detached even if some or all the pages of that memory object are fixed.

,SADMP=DEFAULT
,SADMP=YES
,SADMP=NO
An optional keyword input that specifies whether the memory object is to be captured in a stand-alone dump.

SADMP=DEFAULT
When PAGEFRAMESIZE is not 2G, the memory object should be captured in a stand-alone dump.

When PAGEFRAMESIZE is 2G, the memory object should not be captured in a stand-alone dump unless explicitly requested by the stand-alone dump program.

SADMP=YES
The memory object should be captured in a stand-alone dump.

SADMP=NO
The memory object should not be captured in a stand-alone dump unless explicitly requested by the stand-alone dump program.

Default: SADMP=DEFAULT

,INORIGIN=*inorigin*
An optional 8-byte input parameter that contains the address of the desired storage to be obtained. The address must be on a 1 MB boundary (or on a 2 GB boundary when PAGEFRAMESIZE=2G is specified). No part of the requested area can be in the 0 - 2G area, the 64-bit common area, or the 64-bit shared area. When a specific virtual area is requested (USE2GTO32G, USE2GTO64G, LOCALSYSAREA), the entire INORIGIN requested area must be in that specific virtual area.

To code: Specify the RS-type address, or address in register (2) - (12), of an eight-byte pointer field.

, USE2GTO32G = NO
, USE2GTO32G = YES
An optional keyword input that specifies whether this is an explicit allocation request for 64-bit virtual storage in the 2G to 32G virtual storage area. IBM suggests that you not use this parameter because Java and other language runtimes use it. If there is not enough memory available in this range, the language runtimes could fail to start, or there could be increased memory usage and reduced performance. This parameter relates to usage of the compressed references feature, which is documented in *z/OS User Guide for IBM SDK, Java Technology Edition*.

Check the RCEUSE2GTO32GAREAOK bit in the IARRCE macro to ensure that the system supports this keyword.

USE2GTO32G=NO
The request is not to be satisfied from the 2G to 32G virtual storage area.

USE2GTO32G=YES
The request is to be satisfied from the 2G to 32G virtual storage area. You cannot specify USE2GTO32G=YES with USE2GTO64G=YES. When you specify USE2GTO32G=YES, the system ignores USE2GTO64G=NO.

Default: USE2GTO32G=NO

, USE2GTO64G = NO
, USE2GTO64G = YES
An optional keyword input that specifies whether this is an explicit allocation request for 64-bit virtual storage in the 2G to 64G virtual storage area. IBM suggests that you not use this parameter because Java and other language runtimes use it. If there is not enough memory available in this range, the language runtimes could fail to start, or there could be increased memory usage and reduced performance. This parameter relates to usage of the compressed references feature, which is documented in *z/OS User Guide for IBM SDK, Java Technology Edition*.

Check the RCE_USE2GTO64GENABLE bit in the IARRCE macro to ensure that the system supports this keyword.

USE2GTO64G=NO
The request is not to be satisfied from the 2G to 64G virtual storage area.

USE2GTO32G=YES
The request is to be satisfied from the 2G to 64G virtual storage area. You cannot specify USE2GTO64G=YES with USE2GTO32G=YES. When you specify USE2GTO64G=YES, the system ignores USE2GTO32G=NO.

Default: USE2GTO64G=NO

,EXECUTABLE=SYSTEM_RULES
,EXECUTABLE=YES
,EXECUTABLE=NO
Specifies if code can be executed from the obtained storage on a system that has implemented instruction-execution-protection. The default parameter is ,EXECUTABLE=SYSTEM_RULES.

,EXECUTABLE=SYSTEM_RULES
This parameter indicates that code will obey the current system default.

,EXECUTABLE=YES
This parameter indicates that code is able to be executed from the obtained storage.

,EXECUTABLE=NO
This parameter indicates that code will not be able to be executed from the obtained storage on a system that has implemented instruction-execution-protection. The specification of NO will be ignored when executed on a system that has not implemented instruction-execution-protection or is not running an appropriate level of z/OS.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - GETSHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - GETCOMMON
  - PAGEPROTECT
  - PAGEUNPROTECT

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

Use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=GETSHARED option of IARV64

REQUEST=GETSHARED creates a memory object that can be shared across multiple address spaces.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be running in supervisor state or with PSW key 0-7 to use the following parameters:
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=GETSHARED option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=GETSHARED	
,COND=NO	**Default:** COND=NO
,COND=YES	
,SEGMENTS=*segments*	*segments:* RS-type address or address in register (2) - (12).
,PAGEFRAMESIZE=4K	**Default:** PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=PAGEABLE1MEG	
,KEY=*key*	*key:* RS-type address or address in register (2) - (12).
,KEY=CALLERKEY	**Default:** KEY=CALLERKEY
,FPROT=YES	**Default:** FPROT=YES
,FPROT=NO	
,SENSITIVE=UNKNOWN	**Default:** SENSITIVE=UNKNOWN
,SENSITIVE=YES	
,SENSITIVE=NO	

Syntax	Description
,USERTKN=*usertkn*	*usertkn:* RS-type address or address in register (2) - (12).
,CHANGEACCESS=LOCAL	**Default:** CHANGEACCESS=LOCAL
,CHANGEACCESS=GLOBAL	
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) – (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,ORIGIN=*origin*	*origin:* RS-type address or address in register (2) - (12).
,SADMP=DEFAULT	**Default:** SADMP=DEFAULT
,SADMP=YES	
,SADMP=NO	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3 or 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=GETSHARED
REQUEST=GETSHARED requests that a memory object be created. The memory object is allowed to be shared upon return (through SHAREMEMOBJ). Successful completion of this service creates system interest in the memory object, which must be removed (through DETACH AFFINITY=SYSTEM) before the memory object is freed. Addressability to the memory object is not provided by GETSHARED. Instead, use SHAREMEMOBJ to enable the virtual storage to be referenced. Initial access to the memory object is read/write. A memory object created by GETSHARED is not eligible for PAGEFIX or PAGEUNFIX.

,COND=NO
,COND=YES
An optional keyword input that specifies whether the request is unconditional or conditional. The default is COND=NO.

,COND=NO
The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request will not be abnormally ended for resource unavailability.

Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete but a return code will be set to indicate that the request could not be completed successfully.

,SEGMENTS=*segments*
A required input parameter that specifies the size of storage requested in megabytes. This must be a non-zero value.

The amount of storage requested is not charged against the MEMLIMIT.

To code: Specify the RS-type address, or address in register (2) - (12), of a required doubleword field.

,PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=PAGEABLE1MEG
An optional input parameter that specifies the size of the page frames used to back the virtual storage mapped by the allocated memory object. The default is PAGEFRAMESIZE=4K.

,PAGEFRAMESIZE=4K
The memory object should be backed by 4 kilobyte (4K) page frames.

,PAGEFRAMESIZE=PAGEABLE1MEG
The memory object should be backed by 1 megabyte page frames. If 1 megabyte page frames are not supported or not available, the system will attempt to back the memory object with 4K page frames. Note that you cannot specify CHANGEACCESS=LOCAL with this page frame size.

,KEY=*key*
,KEY=CALLERKEY
An optional input parameter that specifies the storage key to be assigned to the memory object. The key must be in bits 0 - 3 of the specified byte. Bits 4 - 7 are ignored.

If the key is not specified, the storage key of the memory object is the same as the caller's PSW key. The default is KEY=CALLERKEY.

To code: Specify the RS-type address, or address in register (2) - (12), of a one-byte field.

,FPROT=YES
,FPROT=NO
An optional parameter that specifies whether the memory object should be fetch protected. The default is FPROT=YES.

,FPROT=YES
The entire memory object will be fetch protected. A program must have a PSW key that matches the storage key of the memory object (or have PSW key 0) to reference data in the memory object.

,FPROT=NO
The memory object will not be fetch protected.

,SENSITIVE=UNKNOWN
,SENSITIVE=YES
,SENSITIVE=NO
An optional keyword input that specifies whether the memory object will contain data that is potentially sensitive (for instance, private or confidential). The default is SENSITIVE=UNKNOWN.

,SENSITIVE=UNKNOWN
Specifies that the sensitivity of the data is not known. The memory object may or may not contain sensitive data.

,SENSITIVE=YES
Specifies that the memory object contains sensitive data.

,SENSITIVE=NO
Specifies that the memory object does not contain sensitive data.

,USERTKN=*usertkn*
This parameter is the name (RS-type), or address in register (2) - (12), of a required doubleword input that identifies the user token to be associated with the shared memory object. This can be used on a later DETACH invocation to affect all memory objects associated with this value. A single shared memory object may be associated with multiple user tokens via GETSHARED and SHAREMEMOBJ.

For authorized callers, bits 0 - 31 of the 64-bit user token must not all be zero; for non-authorized callers, these bits must all be zero.

Note: The scope of a user token value is the entire z/OS image. It can be associated with shared, common, or private memory objects.

,CHANGEACCESS=LOCAL
,CHANGEACCESS=GLOBAL
An optional parameter that specifies whether the subsequent CHANGEACCESS requests are treated as local or global. The default is CHANGEACCESS=LOCAL.

,CHANGEACCESS=LOCAL
The CHANGEACCESS for this memory object will have local scope. Subsequent CHANGEACCESS requests will change access only for the address space specified by CHANGEACCESS. Note that you cannot specify PAGEFRAMESIZE=PAGEABLE1MEG with this scope.

,CHANGEACCESS=GLOBAL
The CHANGEACCESS for this memory object will have global scope. Subsequent CHANGEACCESS requests will change access for all address spaces sharing the memory object and any new address spaces that will subsequently share it.

Note: Use of GLOBAL may reduce system resources needed to manage the memory object and is encouraged when all spaces will be using the same view.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
This parameter is the name of an optional fullword integer input that indicates the ALET of the address space which will be used to create the memory object.

The only supported values are 0 (primary) and 2 (home). The default value is 0.

,ORIGIN=*origin*
A required output parameter that contains the lowest address of the memory object.

To code: Specify the RS-type address, or address in register (2) - (12), of an eight-byte pointer field.

,SADMP=DEFAULT
,SADMP=YES
,SADMP=NO
An optional keyword input that specifies whether the memory object is to be captured in a stand-alone dump.

SADMP=DEFAULT
When PAGEFRAMESIZE is not 2G, the memory object should be captured in a stand-alone dump.

When PAGEFRAMESIZE is 2G, the memory object should not be captured in a stand-alone dump unless explicitly requested by the stand-alone dump program.

SADMP=YES
The memory object should be captured in a stand-alone dump.

SADMP=NO
The memory object should not be captured in a stand-alone dump unless explicitly requested by the stand-alone dump program.

Default: SADMP=DEFAULT

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3 or 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT

  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3 or 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=GETCOMMON option of IARV64

Use REQUEST=GETCOMMON to create a 64-bit common memory object.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be running in supervisor state and with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts Callers that specify PAGEFRAMESIZE=1M or PAGEFRAMESIZE=MAX must be enabled
Locks:	For enabled callers no requirement. For disabled callers no spin locks higher than the RSM locks can be held
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The REQUEST=GETCOMMON option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=GETCOMMON	
,COND=NO	**Default:** COND=NO
,COND=YES	
,SEGMENTS=*segments*	*segments:* RS-type address or address in register (2) - (12).
,PAGEFRAMESIZE=4K	**Default:** PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1MEG	
,PAGEFRAMESIZE=MAX	
,PAGEFRAMESIZE=PAGEABLE1MEG	
,PAGEFRAMESIZE=DREF1MEG	

Syntax	Description
,TYPE=PAGEABLE	**Default:** TYPE=PAGEABLE when one of the following is specified: • PAGEFRAMESIZE=PAGEABLE1MEG • PAGEFRAMESIZE=4K • PAGEFRAMESIZE=MAX and the memory object is backed with 4 KB-page frames. When PAGEFRAMESIZE=DREF1MEG is specified, the default value is TYPE=DREF. If PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX is specified and the memory object is backed with 1 MB-page frames, the 1 MB-pages backing this memory object are fixed.
,TYPE=DREF	
,UNITS=*units*	*units:* Size of the memory object, which is the number of units specified by UNITSIZE.
,UNITSIZE=1M	Specifies a 1 MB unit size.
,PAGEFRAMESIZE=4K|1M	If UNITSIZE=1M is specified, a PAGEFRAMESIZE of 4K or 1M must be specified. There is no default value.
,TYPE=FIXED	
,TYPE=DREF	
,TYPE=PAGEABLE	
,KEY=*key*	*key:* RS-type address or address in register (2) - (12).
,KEY=CALLERKEY	**Default:** KEY=CALLERKEY
,FPROT=YES	**Default:** FPROT=YES
,FPROT=NO	
,SENSITIVE=UNKNOWN	**Default:** SENSITIVE=UNKNOWN
,SENSITIVE=YES	
,SENSITIVE=NO	
,MOTKNSOURCE=USER	**Default:** MOTKNSOURCE=USER
,MOTKN=*motkn*	*motkn:* RS-type address or address in register (2) – (12).
,MOTKNSOURCE=SYSTEM	
,OUTMOTKN=*outmotkn*	*outmotkn:* RS-type address or address in register (2) – (12).

Syntax	Description
,OWNERCOM=HOME	**Default:** OWNER=HOME
,OWNERCOM=PRIMARY	
,OWNERCOM=SYSTEM	
,OWNERCOM=BYASID	
,OWNERASID=ownerasid	
,OWNERASID=HOME	
,GUARDSIZE=*guardsize*	*guardsize:* RS-type address or address in register (2) - (12).
,GUARDSIZE=0	**Default:** GUARDSIZE=0
,GUARDSIZE64=*guardsize64*	*guardsize64:* RS-type address or address in register (2) - (12).
,GUARDSIZE64=0	**Default:** GUARDSIZE64=0
,GUARDLOC=LOW	**Default:** GUARDLOC=LOW
,GUARDLOC=HIGH	
,DUMP=LIKECSA	**Default:** DUMP=LIKECSA when TYPE=PAGEABLE
,DUMP=LIKESQA	**Default:** DUMP=LIKESQA when TYPE=DREF
,DUMP=NO	
,DUMP=BYOPTIONVALUE	
,OPTIONVALUE=*optionvalue*	**option:** RS-type address or address in register (2) - (12).
,ORIGIN=*origin*	*origin:* RS-type address or address in register (2) - (12).
,DETACHFIXED=NO	**Default:** DETACHFIXED=NO
,DETACHFIXED=YES	
,SADMP=DEFAULT	**Default:** SADMP=DEFAULT
,SADMP=YES	
,SADMP=NO	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).

Syntax	Description
,PLISTVER=IMPLIED_VER	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4, 5	
,MF=S	**Default:** MF=S
,MF=*(L,list addr)*	*(L,list addr):* RS-type address or register (1) - (12).
,MF=*(L,list addr,attr)*	
,MF=*(L,list addr,OD)*	
,MF=*(E,list addr)*	
,MF=*(E,list addr,COMPETE)*	

Parameters

The REQUEST=GETCOMMON option of the IARV64 macro is written as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=GETCOMMON
A required parameter. REQUEST=GETCOMMON creates a 64-bit common memory object.

,COND=NO
,COND=YES
An optional keyword input that specifies whether the request is unconditional or conditional. The default is COND=NO.

,COND=NO
The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request will not be abnormally ended for resource unavailability.

Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete and a return code will be set to indicate that the request could not be completed successfully.

,KEY=*key*
,KEY=CALLERKEY
An optional input parameter that specifies the storage key to be assigned to the memory object. The key must be in bits 0-3 of the specified byte. Bits 4-7 are ignored. Only keys 0-7 can be specified.

If the key is not specified, the storage key of the memory object is the same as the caller's PSW key. The default is CALLERKEY.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,FPROT=YES
,FPROT=NO
An optional keyword parameter that specifies whether the memory object should be fetch protected. The default is FPROT=YES

,FPROT=YES
The entire memory object will be fetch protected. A program must have a PSW key that matches the storage key of the memory object (or have PSW key 0) to reference data in the memory object.

,FPROT=NO
The memory object will not be fetch protected.

,SENSITIVE=UNKNOWN
,SENSITIVE=YES
,SENSITIVE=NO
An optional keyword input that specifies whether the memory object will contain data that is potentially sensitive (for instance, private or confidential). The default is SENSITIVE=UNKNOWN.

,SENSITIVE=UNKNOWN
Specifies that the sensitivity of the data is not known. The memory object may or may not contain sensitive data.

,SENSITIVE=YES
Specifies that the memory object contains sensitive data.

,SENSITIVE=NO
Specifies that the memory object does not contain sensitive data.

,MOTKNSOURCE=USER
,MOTKNSOURCE=SYSTEM
An optional input parameter that indicates the source of the memory object token to be associated with this memory object. The default is USER.

,MOTKNSOURCE=USER
The user provides the memory object token.

,MOTKN=*motkn*
The name of an optional doubleword integer input that identifies the token to be associated with the memory object. This must be a token that was returned by the system on a previous GETCOMMON request by the OUTMOTKN keyword. If you specify no user token, the default is that no user token is supplied to associate this memory object with others.

,MOTKNSOURCE=SYSTEM
The system provides the memory object token.

,OUTMOTKN=*xoutmotkn*
The name of a required doubleword integer output in which the system returns the token associated with this memory object. This token can be used on subsequent GETCOMMON requests as a user-supplied token in order to associate other memory objects with this token. This token can be used on subsequent DETACH requests in order to free all the memory objects that have been associated with this token.

Usage notes of the MOTKNSOURCE parameter on an IARV64 REQUEST(GETCOMMON) request:

- If you want a system-generated token to be returned, invoke:

```
IARV64 REQUEST=GETCOMMON,MOTKNSOURCE=SYSTEM,OUTMOTKN=mytoken
```

- If you want to use the returned token on subsequent IARV64 GETCOMMON requests in order to associate other memory objects with the same token, invoke:

```
IARV64 REQUEST=GETCOMMON,MOTKNSOURCE=USER,MOTKN=mytoken
```

- If you want to use the returned token on a DETACH request in order to detach all memory objects that are associated with that token, invoke:

```
IARV64 REQUEST=DETACH,MATCH=MOTOKEN,MOTKN=mytoken,
       AFFINITY=SYSTEM,V64COMMON=YES
```

,SEGMENTS=*segments*

SEGMENTS and UNITS are mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the size of the requested memory object, in megabytes. This must be a nonzero value. The amount of storage requested is not charged against the MEMLIMIT for the address space making the request.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1MEG
,PAGEFRAMESIZE=MAX
,PAGEFRAMESIZE=DREF1MEG
,PAGEFRAMESIZE=PAGEABLE1MEG

An optional input parameter that specifies the size of the page frames to back the virtual storage mapped by the allocated memory object.

,PAGEFRAMESIZE=4K
The memory object should be backed by 4 KB-page frames. The default value is PAGEFRAMESIZE=4K.

,PAGEFRAMESIZE=1MEG
The memory object should be backed by 1 MB-page frames.

,PAGEFRAMESIZE=MAX
The memory object should be backed by the largest page frame size supported but if the request cannot be backed by the largest frame size due to the availability of large page frames, then the request will backed by 4 KB-page frames. 1 megabyte page frames are backed at allocation time and ca not be paged out to AUX. 4 KB-page frames are backed at first reference and can be paged out to AUX if TYPE=DREF is not specified or can not be paged out to AUX if TYPE=DREF is specified.

,PAGEFRAMESIZE=PAGEABLE1MEG
The memory object is backed by pageable 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.

,PAGEFRAMESIZE=DREF1MEG
The memory object is backed by pageable 1 MB-page frames at first reference, unless none are available. If none are available, the object is backed by 4 KB-page frames.

,TYPE=PAGEABLE
,TYPE=DREF

An optional input parameter that specifies the type of storage that is requested. The default value is TYPE=PAGEABLE when one of the following parameters is specified:

- PAGEFRAMESIZE=PAGEABLE1MEG
- PAGEFRAMESIZE=4K
- PAGEFRAMESIZE=MAX and the memory object is backed with 4 KB-page frames.

The default value is TYPE=DREF when PAGEFRAMESIZE=DREF1MEG is specified. If PAGEFRAMESIZE=1MEG or PAGEFRAMESIZE=MAX is specified and the memory object is backed with 1 MB-page frames, the 1 MB-pages backing this memory object are fixed.

Note:

1. When the memory object is backed by 4 KB-page frames, the 4 KB-pages backing this memory object are pageable if TYPE=DREF is not specified; the 4 KB-pages are fixed if TYPE=DREF is specified. The 4 KB-pages are backed at first reference and can and can only be paged out to AUX if TYPE=DREF is not specified.

2. When the memory object is backed by 1 MB-page frames as a result of PAGEFRAMESIZE=PAGEABLE1MEG or PAGEFRAMESIZE=DREF1MEG being specified, the 1 MB-pages backing this memory object are pageable if PAGEABLE1MEG is specified or fixed if DREF1MEG is specified. Pageable 1 MB-pages are backed at first reference and can be paged out to AUX. DREF 1 MB-pages are backed at first reference and are fixed—they cannot be paged out to AUX.
3. When the memory object is backed by 1 MB-page frames because PAGEFRAMESIZE=1MEG or PAGEFRAMSIZE=MAX has been specified, the 1 MB-pages backing this memory object are fixed. Pages are backed at allocation time and cannot be paged out to AUX.

,TYPE=PAGEABLE
Pages backing this memory object are pageable. Pages are backed at first reference and can be paged out to AUX. virtual address ranges within the memory object can be explicitly fixed after allocation by using the IARV64 REQUEST=PAGEFIX request.

,TYPE=DREF
The memory object is referenced while running disabled. Note that the DREF attribute applies to the entire memory object. Pages are backed in real at first reference. Pages belonging to memory objects with the TYPE=DREF attribute remain in real and are never paged out to AUX.

,UNITS=*units*

UNITS and SEGMENTS are mutually exclusive keys. This set is required; only one key can be specified.

A required input parameter that specifies the size of the memory object as a number of units of the size specified by the UNITSIZE parameter. This must be a nonzero value. The amount of storage requested is not charged against the MEMLIMIT for the address space making the request.

,TYPE=PAGEABLE
,TYPE=DREF
,TYPE=FIXED
A required input parameter that specifies the type of requested storage.

,TYPE=PAGEABLE
Pages backing this memory object are pageable. Pages are backed at first reference and can be paged out to auxiliary storage. Virtual address ranges within the memory object can be explicitly fixed after allocation by using the IARV64 REQUEST=PAGEFIX request. TYPE=PAGEABLE is not valid with PAGEFRAMESIZE=2G.

,TYPE=DREF
Pages are backed in real memory at first reference, unless DREF storage is not available, in which case the program is ABENDed. Once backed, pages belonging to memory objects of TYPE=DREF remain in real storage and are never paged out to auxiliary storage. The memory object can be referenced while running disabled. The DREF attribute applies to the entire memory object. TYPE=DREF is not valid with PAGEFRAMESIZE=2G.

,TYPE=FIXED
Pages are backed in real storage immediately, unless fixed storage is not immediately available, in which case the request fails. Pages belonging to memory objects of TYPE=FIXED remain in real storage and are never be paged out to auxiliary storage. The memory object can be referenced while running disabled. The FIXED attribute applies to the entire memory object when it is allocated. TYPE=FIXED is not valid with PAGEFRAMESIZE=4K.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,UNITSIZE=1M
,UNITSIZE=2G
A required input parameter that specifies the size for the UNITS parameter: either 1 MB or 2 GB.

,UNITSIZE=1M
Specifies that the memory object is in 1 MB units. For example, a request for UNITS=3 with UNITSIZE=1M is a request for three megabytes of virtual storage starting on a 1 MB boundary. When UNITSIZE=1M is specified, one of the following PAGEFRAMESIZE values must also be specified:

PAGEFRAMESIZE=4K
PAGEFRAMESIZE=1M
A required input parameter that specifies the size of the page frames used to back the virtual storage mapped by the allocated memory object.

PAGEFRAMESIZE=4K
Specifies to back the memory object with 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported.

PAGEFRAMESIZE=1M
Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available, the system attempts to back the memory object at a smaller page frame size of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

,UNITSIZE=2G
Specifies that the memory object is in two-gigabyte (2G) units. For example, a request for UNITS=3 with UNITSIZE=2G is a request for six gigabytes of virtual storage starting on a 2 GB boundary.

PAGEFRAMESIZE=4K|1M
A required input parameter that specifies the size of the page frames that back the virtual storage mapped by the allocated memory object.

PAGEFRAMESIZE=4K
Specifies to back the memory object by 4 KB-page frames of the specified TYPE, when TYPE=PAGEABLE or TYPE=DREF is requested. TYPE=FIXED is not supported with this value.

PAGEFRAMESIZE=1M
Specifies to back the memory object by one-megabyte (1 MB) page frames of the specified TYPE. If 1 MB-page frames are not supported or not available when TYPE=PAGEABLE or TYPE=DREF is requested, the system attempts to back the memory object using a smaller page frame size of the specified TYPE. A TYPE=FIXED request fails if there are no available pages in the requested PAGEFRAMESIZE.

,OWNERCOM=HOME
,OWNERCOM=PRIMARY
,OWNERCOM=SYSTEM
,OWNERCOM=BYASID
An optional input parameter that specifies the entity to which the system will assign ownership of the 64-bit common memory object. The system uses this ownership information to track the use of 64-bit common storage for diagnostic purposes. The default is OWNERCOM=HOME.

,OWNERCOM=HOME
The home address space will be assigned as the owner of the 64-bit common memory object.

,OWNERCOM=PRIMARY
The primary address space will be assigned as the owner of the 64-bit common memory object.

,OWNERCOM=SYSTEM
The system (the 64-bit common memory object is not associated with an address space) will be assigned as the owner of the 64-bit memory object.

,OWNER=BYASID
The address space specified by OWNERASID will be assigned as the owner of the 64-bit common memory object.

,OWNERASID=0
,OWNERASID=*ownerasid*
An optional input parameter that specifies the ASID of the address space that will own the 64-bit common memory object for tracking purposes. The default is OWNERASID=0.

OWNERASID=0
This parameter indicates that the system is assigned as the owner of the 64-bit memory object.

OWNERASID=ownerasid
This is the name (RS-Type), or address in register (2)-(12), of an optional halfword input that contains the address space identifier (ASID) to be designated as the owner of the 64-bit common memory object for storage tracking purposes.

,GUARDSIZE=*guardsize*
,GUARDSIZE=0

GUARDSIZE and GUARDSIZE64 are mutually exclusive keys. This set is optional; only one key may be specified. A fullword integer input parameter that indicates the number of megabytes of guard area to be created at the high or low end of the memory object. Guard areas cannot be referenced and when referenced will cause a program check. Guard area does not count against the MEMLIMIT. Guard areas cannot be referenced and, if referenced, will cause a program check.

A guard area can be reduced through CHANGEGUARD CONVERT=FROMGUARD. GUARDSIZE must not be larger than the size of the memory object.

GUARDSIZE must not be larger than the size of the memory object. The default is 0.

Default: 0

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field.

,GUARDSIZE64=*guardsize64*
,GUARDSIZE64=0

GUARDSIZE64 belongs to a set of mutually exclusive keys. This set is optional; only one key may be specified. A doubleword integer input parameter that indicates the number of megabytes of guard area to be created at the high or low end of the memory object. Guard areas cannot be referenced and, if referenced, will cause a program check.

A guard area can be reduced through CHANGEGUARD CONVERT=FROMGUARD. GUARDSIZE64 must not be larger than the size of the memory object.

Default: 0

To code: Specify the RS-type address, or address in register (2) - (12), of a doubleword field.

,GUARDLOC=LOW
,GUARDLOC=HIGH
An optional input parameter that specifies whether the guard location is at the low virtual end of the memory object or the high virtual end.

,GUARDLOC=LOW
The guard areas are created starting from the origin of the memory object, that is, from the low virtual end.

,GUARDLOC=HIGH
The guard areas are created at the end of the memory object, that is, at the high virtual end.

Default: GUARDLOC=LOW

,DUMP=LIKECSA
,DUMP=LIKESQA
,DUMP=NO
,DUMP=BYOPTIONVALUE
An optional input parameter that specifies whether the 64-bit common memory object is included in an SVC dump when CSA or SQA is specified on SDATA. When TYPE=PAGEABLE is specified on IARV64 GETCOMMON the default is DUMP=LIKECSA. When TYPE=DREF is specified on IARV64 GETCOMMON the default is DUMP=LIKESQA.

,DUMP=LIKECSA
The 64-bit common memory object is included in an SVC dump when CSA is specified on SDATA.

,DUMP=LIKESQA
The 64-bit common memory object is included in an SVC dump when SQA is specified on SDATA.

,DUMP=NO
The 64-bit common memory object is not included in an SVC dump when either CSA or SQA is specified on SDATA.

DUMP=BYOPTIONVALUE
The 64-bit common memory object is dumped according to the option specified by the OPTIONVALUE keyword.

,OPTIONVALUE=option
This parameter is the name (RS-Type), or address in register (2) - (12), of a required one-byte integer input that contains one of the following:

- XMFCTRL_XDUMP_NO - (X'01') — this is equivalent to DUMP=NO
- XMFCTRL_XDUMP_LIKESQA - (X'02') — this is equivalent to DUMP=LIKESQA
- XMFCTRL_XDUMP_LIKECSA - (X'03') — this is equivalent to DUMP=LIKECSA

,DETACHFIXED=NO
,DETACHFIXED=YES
An optional input parameter that specifies whether the memory object can be detached when it contains fixed pages at the time of the DETACH request. The default value for DETACHFIXED is NO.

DETACHFIXED=NO
The memory object will not be detached if it has any fixed pages when it is detached.

DETACHFIXED=YES
The memory object will be detached even if some or all the pages of the memory object are fixed.

,ORIGIN=*origin*
A required output parameter that contains the lowest address of the memory object.

Note: When GUARDLOC=LOW is specified, the lowest address will point to a guard area which will cause an ABEND if referenced. For GUARDLOC=LOW the first usable area is the origin plus the size of the guard area.

To code: Specify the RS-type address or address in register (2) - (12) of an eight-byte pointer field.

,SADMP=DEFAULT
,SADMP=YES
,SADMP=NO
An optional keyword input that specifies whether the memory object is to be captured in a stand-alone dump.

SADMP=DEFAULT
When PAGEFRAMESIZE is not 2G, the memory object should be captured in a stand-alone dump.

When PAGEFRAMESIZE is 2G, the memory object should not be captured in a stand-alone dump unless explicitly requested by the stand-alone dump program.

SADMP=YES
The memory object should be captured in a stand-alone dump.

SADMP=NO
The memory object should not be captured in a stand-alone dump unless explicitly requested by the stand-alone dump program.

Default: SADMP=DEFAULT

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field or register (2) - (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are::

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default
- **MAX**,if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - GETSHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:

  - DMAPAGETABLE
- **5**, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4 or 5

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(M,*list addr*,)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
This parameter specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=CHANGEACCESS option of IARV64

REQUEST=CHANGEACCESS requests that the view type for segments within the specified 64-bit shared memory objects can be changed.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be running in supervisor state or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31- or 64-bit.
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=CHANGEACCESS option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=CHANGEACCESS	
,VIEW=READONLY	**Default:** VIEW=READONLY
,VIEW=SHAREDWRITE	
,VIEW=HIDDEN	
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0

Syntax	Description
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3 or 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

IARV64 REQUEST=CHANGEACCESS requests that the view type for segments within the specified 64-bit shared memory objects can be changed.

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=CHANGEACCESS
REQUEST=CHANGEACCESS requests that the type of access to the specified virtual storage be changed. For 64-bit shared memory objects the scope of the change is determined by the choice of LOCAL versus GLOBAL on the IARV64 GETSHARED CHANGEACCESS keyword.

For 64-bit shared memory objects when CHANGEACCESS=LOCAL was specified or defaulted on the REQUEST=GETSHARED, only the address space specified by the ALET is affected.

For 64-bit shared memory objects when the CHANGEACCESS=GLOBAL is specified, all address spaces currently sharing the memory object are affected. Subsequent IARV64 SHAREMEMOBJ requests for this memory object will also be affected by this CHANGEACCESS when CHANGEACCESS=GLOBAL is specified (until the next CHANGEACCESS invocation).

The memory object specified must be a 64-bit shared memory object. For example, it is the result of a GETSHARED invocation

CHANGEACCESS requests for memory objects that are CHANGEACCESS=LOCAL require that the target space have interest in the shared memory object. For example, a SHAREMEMOBJ for the target space must have been done before the CHANGEACCESS request. Memory objects with CHANGEACCESS=GLOBAL support CHANGEACCESS requests without prior SHAREMEMOBJ requests.

,VIEW=READONLY
,VIEW=SHAREDWRITE
,VIEW=HIDDEN
A required input parameter that indicates the accessing mode on the area.

,VIEW=READONLY
This parameter specifies that the area can only be used to read data. Any attempt to alter data by writing onto the area will result in a program check.

,VIEW=SHAREDWRITE
This parameter specifies that the area can be used to read or update data.

,VIEW=HIDDEN
This parameter specifies that the data within the area cannot be accessed until its view type is changed to READONLY or SHAREDWRITE. Any attempt to access a hidden area will result in a program check.

,RANGLIST=*ranglist*
A required input parameter that contains the address of the ranglist. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. Each range list entry contains the following fields:

VSA
VSA denotes the starting virtual address of the data to be acted on. The virtual address specified must be within a memory object returned by GETSHARED (not GETSTOR or GETCOMMON). The value must always be on a segment boundary.

NUMSEGMENTS
NUMSEGMENTS contains the number of segments (megabytes) in the area. The number of segments specified starting with the specified VSA must lie within a single memory object. The length of this field is 8 bytes. A value of 0 is valid for NUMSEGMENTS; it has no special meaning and is treated literally.

To code: Specify the RS-type address, or address in register (2) - (12), of an eight-byte pointer field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space sharing a memory object that will change access to the memory object.

The only supported values are 0 (primary) and 2 (home).

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3 or 4

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3 or 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=CHANGEATTRIBUTE option of IARV64

REQUEST=CHANGEATTRIBUTE requests to change an attribute of storage within one or more memory objects.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be running in supervisor state or with PSW key 0-7 or have a PSW key that matches the storage key of the memory object to be changed for a CHANGEATTRIBUTE request.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31- or 64-bit.
ASC mode:	Primary or access register (AR).

Environmental factor	Requirement
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	A local lock may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None.

Restrictions

Before issuing the IARV64 REQUEST=CHANGEATTRIBUTE macro, it is the caller's responsibility to check that the RCE_SensitiveSupportApplied and RCE_ChangeAttributeApplied bits in the IARRCE macro are set to 1.

This macro supports multiple versions. Some keywords are unique to certain versions. See the PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2 - 13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0 - 1
Used as work registers by the system

2 - 13
Unchanged

14 - 15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the service returns control.

Performance implications

None.

Syntax

The REQUEST=CHANGEATTRIBUTE option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=CHANGEATTRIBUTE	
,RANGLIST=*ranglist*	*ranglist*: RS-type address or address in register (2) - (12)
,SENSITIVE=UNKNOWN	
,SENSITIVE=YES	
,SENSITIVE=NO	
,ALETVALUE=*aletvalue*	*aletvalue*: RS-type address or address in register (2) - (12)
,ALETVALUE=0	**Default**: ALETVALUE=0
,NUMRANGE=*numrange*	*numrange*: RS-type address or address in register (2) - (12)
,NUMRANGE=1	**Default**: NUMRANGE=1
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12), or (15), (GPR15)
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12), or (00), (GPR0)
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=5	The PLISTVER value can be 5 or higher.

Syntax	Description
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr*,*attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are described as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=CHANGEATTRIBUTE
A required parameter that notifies the system that an attribute of the specified range or ranges is to be changed. This request is only supported when the RCE_ChangeAttributeApplied bit in IARRCE is set.

Minimum authorization required: The caller must be running in supervisor state or with PSW key 0-7 or have a PSW key that matches the storage key of the data to be changed. In addition, the caller must be running in supervisor state or with PSW key 0-7 to use the ALETVALUE keyword.

,RANGLIST=*ranglist*
A required input parameter that contains the address of a list of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. Each entry contains the following fields:

VSA
An 8-byte field that denotes the starting virtual address of the data to be changed. The value must be on a page (4K) boundary.

The specified address must be within storage obtained by one of the following services:

- IARV64 GETSTOR
- IARV64 GETCOMMON
- IARV64 GETSHARED

Changing the SENSITIVE attribute is not allowed for memory objects obtained by the following services:

- IARST64
- IARCP64

AMOUNT
An 8-byte field that contains the number of 4K pages to be changed. The specified number of pages, starting with the VSA, must lie within a single memory object.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-byte pointer field.

,SENSITIVE=[UNKNOWN|YES|NO]
A required parameter that specifies the data type attribute to be assigned to the pages. Marking data as sensitive allows installations to easily remove it from certain system diagnostics.

,SENSITIVE=UNKNOWN
Specifies that the data type of the pages is not known. The pages may or may not contain sensitive data.

,SENSITIVE=YES
Specifies that the pages contain sensitive data.

,SENSITIVE=NO
Specifies that the pages do not contain sensitive data.

,ALETVALUE=[*aletvalue*|0]
An optional input parameter that indicates the ALET of the address space that owns or has access to the memory object in which the storage attribute is to be changed.

The only supported values are 0 (primary) and 2 (home). The default is 0.

The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0 - 7.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,NUMRANGE=[*numrange*|1]
An optional input parameter that specifies the number of entries in the supplied range list.

The specified value must be less than or equal to 16. The default is 1.

When multiple range list entries are provided, they can be a mixture of private, common, and shared memory objects.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12), or (15), (GPR15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12), or (00), (GPR0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=5
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms.

The SENSITIVE keyword requires a version level of 5 or higher.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **5**, supports both the following parameter and the parameters from lower versions:
  - SENSITIVE

To code: Specify one of the following:

- **IMPLIED_VERSION**

- **MAX**
- A decimal value of 5 or higher

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=CHANGEGUARD option of IARV64

IARV64 REQUEST=CHANGEGUARD requests that a specified amount of a private or common memory object be changed from the guard area to the usable area or vice versa.

IARV64 REQUEST=CHANGEGUARD only applies to 64-bit private or common memory objects. If a 64-bit memory object backed by a 2G frame (PAGEFRAMESIZE=2G) or a 64-bit shared memory object is specified on the request, a DC2 abend with reason code X'xx0058xx' is issued.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8 - 15.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN **Note:** The problem state caller running in PSW key 8 - 15 can use CHANGEGUARD only when the primary address space is the home address space.
AMODE:	31- or 64-bit

Environmental factor	Requirement
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	A local lock may be held, subject to the following limitation: When a local lock is held for a CHANGEGUARD request, the lock must be for the address space specified (or is set as the default) by the input ALETVALUE.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=CHANGEGUARD option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=CHANGEGUARD	
,CONVERT=TOGUARD	
,CONVERT=FROMGUARD	
,MEMOBJSTART=*memobjstart*	*memobjstart:* RS-type address or address in register (2) - (12).
,CONVERTSTART=*convertstart*	*convertstart:* RS-type address or address in register (2) - (12).
,CONVERTSIZE=*convertsize*	*convertsize:* RS-type address or address in register (2) - (12).
,CONVERTSIZE64=*convertsize64*	*convertsize64:* RS-type address or address in register (2) - (12).
,COND=NO	**Default:** COND=NO
,COND=YES	
,ALETVALUE=0	**Default:** ALETVALUE=0
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).

Syntax	Description
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=CHANGEGUARD
A required parameter. REQUEST=CHANGEGUARD changes the amount of guard area in the specified memory object. It changes part of the memory object from a guard area to a usable area, or vice versa.

If the CHANGEGUARD service finds a PAGEFIXed area in the area to be converted into a guard area, the caller will be abnormally ended. If a request is made to guard a guard area or to unguard an area that is not guarded a return code 04 will be issued.

For a problem state program running in PSW key (8 - 15), the PSW key of the caller must match the storage key of the memory object and the memory object must be owned by one of the following:

- The calling task
- The job step task
- An ancestor task up through the job step task

,CONVERT=TOGUARD
,CONVERT=FROMGUARD
A required parameter that specifies whether to add or remove guard areas.

,CONVERT=TOGUARD
Convert the specified amount of usable areas to the guard areas. The sensitive state of the guard area will be reset to the memory object's original state and the data in the converted areas will be released. This operation reduces the amount of virtual storage that contributes toward the MEMLIMIT for the address space identified by ALETVALUE. If CONVERTSTART is used then a guard area is created from a usable area starting with the address specified continuing for the number of segments specified by CONVERTSIZE. If CONVERTSTART is not used when GUARDLOC=LOW was specified on the GETSTOR request, the first usable virtual address space in the memory object is increased. If CONVERTSTART is not used when GUARDLOC=HIGH was

specified on the GETSTOR request, the last usable virtual address space in the memory object is decreased.

,CONVERT=FROMGUARD
Convert the specified amount of guard area to be usable area. Any previously guarded pages that were converted as part of this request will appear as pages of zeros. Any pages that were already within a usable area will be unchanged. This operation increases the amount of area that contributes toward the MEMLIMIT for the address space designated by ALETVALUE.

If CONVERTSTART is used then a usable area is created from a guard area starting with the address specified continuing for the number of segments specified by CONVERTSIZE. If CONVERTSTART is not used when GUARDLOC=LOW is specified, the first usable virtual address space in the memory object is decreased. If CONVERTSTART is not used when GUARDLOC=HIGH is specified, the last usable virtual address space in the memory object is increased.

,MEMOBJSTART=*memobjstart*
MEMOBJSTART and CONVERTSTART are a set of mutually exclusive keys. This set is required; only one keyword must be specified. An input parameter that belongs to required a set of mutually exclusive keys. It is the name (RS-type), or address in register (2) - (12), of an eight-byte input that contains the address of the first byte in the memory object.

,CONVERTSTART=*convertstart*
CONVERSTART and MEMOBJSTART are a set of mutually exclusive keys. This set is required; only one keyword must be specified. An input parameter that belongs to a required set of mutually exclusive keys. CONVERTSTART specifies the address to add a guard area (continuing to the virtual address specified by adding the bytes defined in CONVERTSIZE to CONVERTSTART minus one) when CONVERT(TOGUARD) is requested, and specifies the address to remove from the guard area (continuing to the virtual address space specified by adding the bytes defined by CONVERTSIZE to CONVERTSTART minus one) when CONVERT(FROMGUARD) is requested.

Two contiguous guard areas will be consolidated into one contiguous guard area whenever possible. For example, if the guard area that was defined when the memory object was created is contiguous with a guard area created using CONVERTSTART, then the two guard areas are combined into one.

CONVERTSTART is not permitted for high virtual common memory objects; use MEMOBJSTART instead.

Specifying MEMOBJSTART will change the guard area only at the beginning or the end of the memory object. Whether the guard area is at the beginning or the end is specified by the GUARDLOC=[HIGH|LOW] parameter on the IARV64 REQUEST=GETSTOR or REQUEST=GETCOMMON request.

IBM recommends that if CONVERTSTART is used to manage the guard areas within a memory object that all REQUEST=CHANGEGUARD use CONVERTSTART.

To code: Specify the RS-type address, or address in register (2) - (12), of an eight-byte pointer field.

,CONVERTSIZE=*convertsize*
CONVERTSIZE and CONVERTSIZE64 are a set of mutually exclusive keys. This set is required; only one key must be specified. A fullword integer input parameter, that indicates the number of contiguous megabytes that should be removed from the guard area (FROMGUARD) or that should be changed to being part of the guard area (TOGUARD).

For CONVERT=TOGUARD and MEMOBJSTART, CONVERTSIZE or CONVERTSIZE64 must not be larger than the number of usable pages in the memory object to allow successful completion. For CONVERT=FROMGUARD, CONVERTSIZE must not be larger than the number of remaining pages in the default guard area of the memory object to allow successful completion.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field.

,CONVERTSIZE64=*convertsize64*
CONVERTSIZE64 and CONVERTSIZE are a set of mutually exclusive keys. This set is required; only one key must be specified. A doubleword integer input parameter, that indicates the number of contiguous megabytes that should be removed from the guard area (FROMGUARD) or that should be changed to being part of the guard area (TOGUARD).

For CONVERT=TOGUARD and MEMOBJSTART, CONVERTSIZE or CONVERTSIZE64 must not be larger than the number of usable pages in the memory object to allow successful completion. For CONVERT=FROMGUARD, CONVERTSIZE must not be larger than the number of remaining pages in the default guard area of the memory object to allow successful completion.

To code: Specify the RS-type address, or address in register (2) - (12), of a doubleword field.

,COND=NO
,COND=YES
An optional input parameter that specifies whether the request is unconditional or conditional. The default is COND=NO.

,COND=NO
The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request will not be abnormally ended when a MEMLIMIT violation occurs.

Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. When you code COND=YES and there is insufficient storage to satisfy the request, instead of the request being abnormally ended, the request will complete but a return code will be set to indicate that the request could not be completed successfully.

,ALETVALUE=0
,ALETVALUE=*aletvalue*
An optional input parameter that indicates the ALET of the address space in which the memory object resides.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0 - 7. The default is 0.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.

- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60 character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=COUNTPAGES option of IARV64

REQUEST=COUNTPAGES requests the count of the number of 4K pages currently in use in real storage, on auxiliary storage, and in both real storage and on auxiliary storage to back the input high virtual storage ranges.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or with PSW key 0-7. The caller must be running in supervisor state or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts.

Environmental factor	Requirement
Locks:	A local lock may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=COUNTPAGES option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.
REQUEST=COUNTPAGES	
,COND=NO	**Default:** COND=NO
,COND=YES	
,V64LISTPTR=*v64listptr*	*v64listptr:* RS-type address or address in register (2) - (12).
,V64LISTLENGTH=*v64listlength*	*v64listlength:* RS-type address or address in register (2) - (12).
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12). **Default:** NUMRANGE=1
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12). **Default:** ALETVALUE=0
,UNLOCKED=NO	**Default:** NO
,UNLOCKED=YES	
,DISCARDPAGES=NO	**Default:** NO
,DISCARDPAGES=YES	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (0) or (2) - (12).

Syntax	Description
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=COUNTPAGES
A required parameter. REQUEST=COUNTPAGES requests the count of the number of 4K pages currently in use in real storage, on auxiliary storage, and in both real storage and on auxiliary storage to back the input high virtual storage ranges. The counts are returned as grand totals for all pages within the ranges requested. If a page is in real storage and on auxiliary storage, it will be counted in real storage, on auxiliary storage, and in both real storage and on auxiliary storage. See IAXV64WA for a description of the output area.

,COND=NO
,COND=YES
An optional parameter that specifies whether the request is unconditional or conditional. The default is COND=NO.

,COND=NO
The request is unconditional. The request is abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request is not abnormally ended for resource unavailability.

Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work.

,V64LISTPTR=*v64listptr*
A required input parameter that contains the address of the work area that will contain the results of the COUNTPAGES request. The work area must be in fixed storage addressable from the address space for which the request is made and must be initialized to zero by the caller.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,V64LISTLENGTH=*v64listlength*
A required input parameter that specifies the length of the work area that contains the results of the COUNTPAGES request. The work area must be at least 64 bytes long.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,RANGLIST=*ranglist*
A required input parameter that contains the address of a range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry is as follows:

VSA
Denotes the starting virtual address of the data to be acted on. The virtual address must be on a page boundary and within a memory object returned by GETSHARED, GETSTOR or GETCOMMON.

The length of this field is 8 bytes.

NUMPAGES
Contains the number of 4K pages to be acted on. The number of pages specified, starting with the specified VSA, must lie within a single memory object.

The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of a 8-byte pointer field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list up to 16. The default is NUMRANGE=1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space that will access the memory object.

The only supported values are 0 (primary) and 2 (home).

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,UNLOCKED=NO
,UNLOCKED=YES
An optional parameter that specifies whether the request is to be performed without RSM serialization. This should only be used when the memory objects to be counted will not be modified or detached and an exact count is not needed. These counts will not be accurate because RSM processing can be modifying frame allocations. The default is UNLOCKED=NO.

- **UNLOCKED=NO**: The request will use RSM serialization. This is the recommended option.
- **UNLOCKED=YES**: The request will not hold RSM locks during processing.

,DISCARDPAGES=NO
,DISCARDPAGES=YES

An optional parameter that specifies whether the request is to also count the number of pages that were discarded with *keepreal=yes* and not referenced again since the REQUEST=DISCARDDATA. The default is DISCARDPAGES=NO.

- **DISCARDPAGES=NO**: The request will not include information about discarded pages.
- **DISCARDPAGES=YES**: The request will include discarded pages.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12), (15), (GPR15), (REG15), or (R15)

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from version 0, 1 and 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2 or 3.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=DETACH option of IARV64

REQUEST=DETACH allows you to free one or more memory objects.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15. The caller must be running in supervisor state or with PSW key 0-7 to use the following parameters: • AFFINITY=SYSTEM • OWNER=NO
Dispatchable unit mode:	Task or SRB

Environmental factor	Requirement
Cross memory mode:	Any PASN, any HASN, any SASN **Note:** Note that problem state caller running in PSW key 8-15 can use DETACH only when the primary address space is the home address space.
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	A local lock can be held, subject to the following limitation: When a local lock is held for a request for non-shared memory objects, the lock must be for the address space specified (or defaulted) by the input ALETVALUE.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is nonzero

1
Used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
Return code.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=DETACH option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=DETACH	
,MATCH=SINGLE	**Default:** MATCH=SINGLE
,MATCH=MOTOKEN	
,MATCH=USERTOKEN	
,MEMOBJSTART=*memobjstart*	*memobjstart:* RS-type address or address in register (2) - (12).
,MOTKN=*motkn*	*motkn:* RS-type address or address in register (2) – (12).
,USERTKN=*usertkn*	*usertkn:* RS-type address or address in register (2) - (12).
,USERTKN=NO_USERTKN	**Default:** USERTKN=NO_USERTKN
,MOTKNCREATOR=USER	**Default:** MOTKNCREATOR=USER
,MOTKNCREATOR=SYSTEM	
,AFFINITY=LOCAL	**Default:** AFFINITY=LOCAL

Syntax	Description
,AFFINITY=SYSTEM	
,OWNER=YES	**Default:** OWNER=YES
,OWNER=NO	
,TTOKEN=*ttoken*	*ttoken:* RS-type address or address in register (2) - (12).
,TTOKEN=NO_TTOKEN	**Default:** TTOKEN=NO_TTOKEN
,V64COMMON=NO	**Default:** V64COMMON=NO
,V64COMMON=YES	
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,COND=NO	**Default:** COND=NO
,COND=YES	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3 or 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=DETACH
A required parameter. REQUEST=DETACH frees one or more memory objects. Problem state programs running in PSW key (8-15) can use this function only when the primary address space is the home address space, and can affect only a memory object that is created using GETSTOR CONTROL=UNAUTH. If a problem state program running in PSW key (8–15) tries to free a memory object created with CONTROL=AUTH, the system does not free the memory object and an ABEND will be issued.

A memory object can be affected by DETACH when MATCH=SINGLE is specified without MOTKN/USERTKN. Other invocations of DETACH will affect nonshared memory objects only when a matching user token is passed.

A shared memory object can be affected by DETACH only when a matching user token is passed.

When DETACH MATCH=SINGLE AFFINITY=LOCAL USERTKN is specified against a shared memory object, the shared interest will be removed from the address space designated by ALETVALUE provided the usertoken passed still represents current shared interest by the space.

1. If this address space has no further shared interest in the memory object, then DETACH will also remove addressability for the address space identified by the input ALETVALUE.
2. When the last address space has surrendered its use of a given shared memory object and the system interest has been removed (through DETACH AFFINITY=SYSTEM) the memory object will be freed.

When DETACH MATCH=USERTOKEN AFFINITY=LOCAL is specified and the input user token matches the usertoken provided for a given memory object created through GETSTOR MOTKN, that memory object is freed. If the memory object was created through GETSHARED and the input user token represents current shared interest by the address space, then that interest will be removed. The same two observations as in the prior list apply.

When DETACH MATCH=USERTOKEN AFFINITY=SYSTEM is specified, only shared memory objects are affected. When the input user token matches the system interest, the system interest will be removed. If there is no remaining local interest, then the shared memory object is freed.

All I/O into each memory object specified must be complete before the DETACH is requested. If the DETACH service finds a PAGEFIXed page in the memory object, the memory object will not be freed. Any prior pages will have indeterminate data and the caller will be abnormally ended.

If detaching a 64 bit common memory object, specify AFFINITY=SYSTEM,V64COMMON=YES.

,MATCH=SINGLE
,MATCH=MOTOKEN
,MATCH=USERTOKEN
An optional parameter that indicates which memory objects are to be freed. The default is MATCH=SINGLE.

,MATCH=SINGLE
Specifies that the input contains MEMOBJSTART for a single memory object.

,MATCH=MOTOKEN
Specifies that the input contains a memory object token that was passed to GETSTOR, GETSHARED, or SHAREMEMOBJ. Memory objects not associated with a memory object token are not affected. Such objects would have to have been created using GETSTOR without MOTKN/USERTKN. If MATCH=MOTOKEN or MATCH =USERTOKEN, COND=YES, and no matching memory object token exists, the system returns a return code instead of abnormally ending the caller.

For nonshared memory objects, all memory objects associated with this memory object token are freed unless it is a problem state program with PSW key 8-15 trying to free a memory object created with CONTROL=AUTH.

For shared memory objects, when AFFINITY=LOCAL is given, the shared interest in memory objects associated with this memory object token is to be removed (for the ALET specified through ALETVALUE). If a given shared memory object no longer has outstanding shared interest, then it will be freed.

For shared memory objects, when AFFINITY=SYSTEM is given, the system interest in memory objects associated with this memory object token is to be freed. If a specified shared memory object no longer has outstanding shared interest, then it will be freed.

If the system encounters an error in processing a qualifying memory object, for example, an unexpected pagefixed page, then the processing ends. The system does not process that page or any further pages or memory objects and abnormally ends the caller.

,MATCH=USERTOKEN
This is a synonym for MATCH=MOTOKEN.

,MEMOBJSTART=*memobjstart*
When MATCH=SINGLE is specified, a required input parameter that contains the address of the first byte in the memory object.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-byte pointer field.

,MOTKN=*motokn*
An optional input parameter that identifies the memory object token to uniquely identify the memory object, as previously passed to GETSTOR, GETSHARED or SHAREMEMOBJ, or the token that was generated by the system on a GETCOMMON or GETSTOR request.

Each shared memory object can be associated with multiple memory object tokens. For AFFINITY=LOCAL, the shared interest in a shared memory object associated with this memory object token is to be removed for the address space. For AFFINITY=SYSTEM, the shared interest created by GETSHARED is to be removed or the 64-bit common memory object is to be freed. For either specification of AFFINITY, when a given shared memory object no longer has outstanding shared interest, it is freed.

When the memory object is not associated with the input token value, it is not processed.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field, or specify a literal decimal value.

,USERTKN=*usertkn*
,USERTKN=NO_USERTKN
A parameter that is similar to MOTKN. Unlike MOTKN, a USERTKN is always presumed to be user-created. The default is NO_USERTKN. It is suggested that you use MOTKN rather than USERTKN. MOTKN=xxx,MOTKNCREATOR=USER is equivalent to USERTKN=xxx.

The default can be used only for memory objects created by GETSTOR. When the memory object is created by GETSHARED, it is necessary to specify the memory object token to uniquely identify which shared interest is to be freed.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field, or specify a literal decimal value.

,MOTKNCREATOR=USER
,MOTKNCREATOR=SYSTEM
When MOTKN=*motokn* is specified, an optional parameter that indicates who created the memory object token.

,MOTKNCREATOR=USER
The memory object token is user-created.

,MOTKNCREATOR=SYSTEM
The memory object token is system-created.

The default is MOTKNCREATOR=USER.

,AFFINITY=LOCAL
,AFFINITY=SYSTEM
An optional input parameter that identifies whether local or system affinity for the memory object will be affected. The default is AFFINITY=LOCAL.

,AFFINITY=LOCAL
Local affinity to the memory object is to be affected, the interest in the memory object defined by the input ALETVALUE and memory object token. Nonshared memory objects are affected by AFFINITY=LOCAL.

Shared memory objects for which an appropriate SHAREMEMOBJ has been done by the address space defined by the input ALETVALUE will also be affected by AFFINITY=LOCAL.

Do not use AFFINITY=LOCAL when detaching a 64 bit common memory object.

AFFINITY=SYSTEM
System affinity to the shared or 64-bit common memory object will be affected. Specify AFFINITY=SYSTEM,V64COMMON=YES when detaching a 64 bit common memory object.

AFFINITY=SYSTEM can be used only by callers running in supervisor state of with PSW key 0–7.

,OWNER=YES
,OWNER=NO
When AFFINITY=LOCAL is specified, an optional keyword input that specifies whether the system will check if the ttoken provided or the task of the caller matches the ttoken associated with the memory object when it was created (only relevant for memory objects created through GETSTOR not GETSHARED). The default is OWNER=YES.

,OWNER=YES
The task which owns the memory object must match the current task or the ttoken provided.

OWNER=NO
The task, which is freeing the memory object does not have to be the owner of the memory object. NO can be used only by programs running in supervisor state or with PSW key 0-7.

,TTOKEN=*ttoken*
,TTOKEN=NO_TTOKEN
When OWNER=YES and AFFINITY=LOCAL are specified, an optional input parameter that identifies the task that owns the memory object. The TTOKEN is returned by the TCBTOKEN macro.

If TTOKEN is not specified, the task issuing the DETACH request must be the owner of the memory object.

The task identified by the TTOKEN must be in the address space specified or defaulted by the ALETVALUE keyword.

When the TTOKEN parameter is used by problem state programs with PSW key 8-15, the target task must represent the calling task OR the jobstep task for the calling task OR the mother task. However, the mother task may not be given when the calling task is itself a jobstep task.

If the TTOKEN parameter is not specified, and the caller is a TCB, the currently dispatched task must be the owner of the memory object. When OWNER=YES is specified by an SRB, the caller will be abnormally ended if the TTOKEN value is not supplied. The default is NO_TTOKEN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,V64COMMON=NO
,V64COMMON=YES
When AFFINITY=SYSTEM is specified, an optional input parameter that indicates whether this is memory object is a 64-bit common memory object. The default is V64COMMON=NO.

,V64COMMON=NO
This is not a 64-bit common memory object.

,V64COMMON=YES
This is a 64-bit common memory object.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space of the memory object to be freed.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by programs running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,COND=NO
,COND=YES
An optional keyword input that specifies whether the request is unconditional or conditional. The default is COND=NO.

,COND=NO
The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request will not be abnormally ended for resource unavailability.

Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally ends the active unit of work. When you code COND=YES, instead of the request being abnormally ended, the request will complete but a return code will be set to indicate that the request could not be completed successfully. In all cases, the request will be abnormally ended for invalid requests, including violation of environmental restrictions.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify register 15, the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or register (15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3 or 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64

  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameters and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3 or 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the

parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=DISCARDDATA option of IARV64

REQUEST=DISCARDDATA allows you to discard data within physical pages of one or more memory objects.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15. The caller must be running in supervisor state or with PSW key 0-7 or have a PSW key that matches the storage key of the memory object to be cleared by DISCARDDATA.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	When a local lock is held for a request (GETSTOR, SHAREMEMOBJ, DETACH, or DISCARDDATA) the lock must be for the address space specified (or defaulted) by the input ALETVALUE.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=DISCARDDATA option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.

Syntax	Description
REQUEST=DISCARDDATA	
,KEEPREAL=YES	**Default:** KEEPREAL=YES
,KEEPREAL=NO	
,CLEAR=YES	**Default:** CLEAR=YES
,CLEAR=NO	
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=DISCARDDATA
A required parameter. REQUEST=DISCARDDATA discards the data within the specified ranges.

For shared memory objects, the address space specified by the ALET or the default must have access to the memory object before issuing the DISCARDDATA (must have issued the IARV64 SHAREMEMOBJ prior to issuing the request).

Areas of the memory object that are PAGEFIXed, or are guard areas in the address space identified by the input ALET will not be discarded. If the DISCARDDATA service finds a PAGEFIXed, hidden, read-only, or guard area in the area to be discarded, the caller will be abnormally ended. However, any prior pages processed will have data in an indeterminate state when CLEAR=NO is used, and KEEPREAL=YES is also used or set as the default.

The caller must be in supervisor state or have PSW key 0-7 or have a PSW key that matches the storage key of the memory object to be cleared.

,KEEPREAL=YES
,KEEPREAL=NO
An optional parameter that specifies whether the real frames backing the pages to be discarded are to be freed or not. The default is KEEPREAL=YES.

,KEEPREAL=YES
The real frames backing the pages to be discarded are not to be freed unless there is shortage in real storage.

,KEEPREAL=NO
The real frames backing the pages to be discarded are to be freed. In this case, the CLEAR keyword value is ignored.

,CLEAR=YES
,CLEAR=NO
An optional parameter that specifies whether the data in the range should become binary zeros. The default is CLEAR=YES.

,CLEAR=YES
The data will become binary zeros.

,CLEAR=NO
The data will be indeterminate.

,RANGLIST=*ranglist*
A required input parameter, of a range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
Denotes the starting address of the data to be acted on.

The address specified must be within a memory object returned by GETSTOR, GETSHARED, or GETCOMMON.

The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.

Note: If the starting address is backed by pageable 1 MB-page frames, specify a value on a physical segment boundary. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

NUMPAGES
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

Note: If the range includes addresses backed by pageable 1 MB-page frames, specify a number of 4 KB-pages that is a multiple of 256. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space owning or with access to the memory object in which the virtual storage data is to be discarded.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE

  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=LIST option of IARV64

REQUEST=LIST requests a list of objects be provided to the caller.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15. The caller must be running in supervisor state or with PSW key 0-7 to use the following parameters: • GETSTOR – KEY – CONTROL=AUTH – ALETVALUE – PAGEFRAMESIZE=1M/MAX • V64SHARED • DETACH – AFFINITY=SYSTEM – OWNER=NO • PAGEFIX • PAGEUNFIX • LIST • SHAREMEMOBJ • CHANGEACCESS The caller must be running in supervisor state or with PSW key 0-7 or have a PSW key that matches the storage key of the memory object to be cleared by DISCARDDATA. The caller must be running in supervisor state or with PSW key 0-7 to DETACH a memory object owned by another task.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN. **Note:** The problem state caller running in PSW key 8-15 can use GETSTOR/DETACH only when the primary address space is the home address space.
AMODE:	31- or 64-bit.
ASC mode:	Primary or access register (AR)

Environmental factor	Requirement
Interrupt status:	Enabled for I/O and external interrupts
Locks:	A local lock may be held, subject to the following limitation: When a local lock is held for a requests (GETSTOR, SHAREMEMOBJ, DETACH, or DISCARDDATA) for non-shared memory objects, the lock must be for the address space specified (or defaulted) by the input ALETVALUE.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=LIST option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.
REQUEST=LIST	
,V64LISTPTR=*v64listptr*	*v64listptr:* RS-type address or address in register (2) - (12).
,V64LISTLENGTH=*v64listlength*	*v64listlength:* RS-type address or address in register (2) - (12).
,V64SHARED=NO	**Default:** V64SHARED=NO
,V64SHARED=YES	
,V64COMMON=NO	**Default:** V64COMMON=NO
,V64COMMON=YES	
,TRACKINFO=NO	**Default:** TRACKINFO=NO
,TRACKINFO=YES	
,V64SELECT=NO	**Default:** V64SELECT=NO
,V64SELECT=YES	
,USERTOKEN=usertoken	**Default:** USERTOKEN=NO_USERTKN
,USERTOKEN=*NO_USERTOKEN*	
,SVCDUMPRGN=YES	**Default:** SVCDUMPRGN=YES
,SVCDUMPRGN=NO	

Syntax	Description
,SVCDUMPRGN=ALL	
,DUMP=ALL	**Default:** DUMP=ALL
,DUMP=LIKECSA	
,DUMP=LIKESQA	
,DUMP=LIKERGN	
,DUMPPROTOCOL=NO	**Default:** DUMPPROTOCOL=NO
,ORDER=ASCENDING	**Default:** ORDER=ASCENDING
,ORDER=DUMPPRIORITY	
,DUMPPROTOCOL=YES	
,OWNERCOM=ALL	**Default:** OWNERCOM=ALL
,OWNERCOM=HOME	
,OWNERCOM=PRIMARY	
,OWNERCOM=SYSTEM	
,OWNERCOM=BYASID	
,OWNERASID= *ownerasid*	
,OWNERASID=ALL	**Default:** OWNERASID=ALL
,PAGEFRAMESIZE=All	**Default:** PAGEFRAMESIZE=All
,PAGEFRAMESIZE=4K	
,PAGEFRAMESIZE=1MEG	
,OWNERJOBNAME=*ownerjobname*	*ownerjobname:* RS-type address or address in register (2) - (12).
,OWNERJOBNAME=All	**Default:** OWNERJOBNAME=ALL
,ATTRIBUTE=DEFS	**Default:** ATTRIBUTE=DEFS
,ATTRIBUTE=NOTOWNERGONE	
,ATTRIBUTE=OWNERGONE	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	

Syntax	Description
,PLISTVER=0, 1, 2, 3, 4, 5	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=LIST
A required parameter. REQUEST=LIST provides information about memory objects.

The information is returned in a work area that you specify, in a format described in IAXV64WA. Information includes starting address, ending address, storage key and flags indicating if the memory object is shared or if it contains multiple guard areas. (For more information about IAXV64WA, see *z/OS MVS Data Areas Volume 2 (IAX - ISG)*.)

The following information can be requested:

- Memory objects for the entire address space
- Memory objects in the entire address space that have been marked SVCDUMPRGN=YES
- Memory objects in the entire address space that have a specific SVCDUMPRGN attribute
- Memory objects in the entire address space that have a specific PAGEFRAMESIZE attribute
- Shared memory objects for the entire system
- 64-bit common memory object for the entire system.

,V64LISTPTR=*v64listptr*
A required input parameter that contains the address that specifies the address of the work area which contains the results of the list request. This work area must be in fixed storage addressable from the address space for which the LIST request is made, and must be initialized to zero by the caller.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,V64LISTLENGTH=*v64listlength*
A required input parameter that specifies the length of the work area which contains the results of the list request. The work area must be at least 108 bytes long when TRACKINFO=YES or 64 bytes long when TRACKINFO=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,V64SHARED=NO
,V64SHARED=YES
An optional input parameter that specifies whether the list of memory objects returned is for private memory objects, which the current primary space owns as well as shared memory objects connected

to the current primary address space, or a list of all shared memory objects defined in the system through GETSHARED. The default is V64SHARED=NO.

,V64SHARED=NO
The list of memory objects returned for the current primary address space includes private memory objects that are defined for the private area through an IARV64 GETSTOR and shared memory objects connected to the current primary address space through an IARV64 SHAREMEMOBJ.

,V64SHARED=YES
The list is of shared memory objects defined for the system through GETSHARED.

,V64COMMON=NO
,V64COMMON=YES
An optional parameter that specifies whether the list of memory object returned is for the current primary address space, or a list of all 64-bit common memory objects allocated in the system via an IARV64 REQUEST=GETCOMMON. The default is V64COMMON=NO.

,V64COMMON=NO
The list of memory objects returned for the current primary address space includes private memory objects (which are defined for the private area via an IARV64 REQUEST=GETSTOR), shared memory objects (connected to the current primary address space via an IARV64 REQUEST=SHAREMEMOBJ).

,V64COMMON=YES
The list of memory objects returned contains all 64-bit common memory objects defined in the system via IARV64 REQUEST=GETCOMMON.

,V64SELECT=NO
,V64SELECT=YES

An optional parameter that specifies whether the list request is for all allocated memory objects or for a subset of the allocated memory objects The default is V64SELECT=NO.

,V64SELECT=NO
The request is for all allocated memory objects. No additional selection criteria apply to the list of memory objects returned.

,V64SELECT=YES
The request is for a subset of the allocated memory objects. Only memory objects that meet all the selection criteria is returned. If a selection criteria keyword is not specified, its default will apply. If no memory object meets the selection criteria, no object will be returned.

,TRACKINFO=NO
,TRACKINFO=YES
When V64COMMON=YES is specified, an optional parameter that specifies if the common memory object tracking information should be returned. Default is TRACKINFO=NO.

,TRACKINFO=NO
Common memory object tracking information will not be returned.

,TRACKINFO=YES
Common memory object tracking information will be returned. See IAXV64WA in *z/OS MVS Data Areas Volume 2 (IAX - ISG)* for output format.

,USERTOKEN=NO_USERTKN
,USERTOKEN=*xusertkn*
When V64SELECT=YES is specified, an optional parameter that specifies whether additional selection criteria based on user token is applied to the set of memory object descriptions returned by the LIST request. The default is NO_USERTKN.

,USERTOKEN=NO_USERTKN
The memory objects returned are not filtered based on USERTKN. All memory objects, regardless of what the USERTKN specification was on the IARV64 GETSTOR or IARV64 SHAREMEMOBJ request for V64SHARED=NO or IARV64 GETSHARED request for V64SHARED=YES, are included in the set of memory objects returned.

,USERTOKEN=*xusertkn*
When V64SHARED=NO is specified, memory objects in the current primary address space that have a matching user token specified on the IARV64 GETSTOR or IARV64 SHAREMEMOBJ request are included in the set of memory objects returned. When V64SHARED=YES is specified, shared memory objects defined in the system that have a matching user token specified on the IARV64 GETSHARED request are included in the set of memory objects returned.

,SVCDUMPRGN=YES
,SVCDUMPRGN=NO
,SVCDUMPRGN=ALL
When V64SELECT=YES is specified, an optional parameter that specifies whether the memory object should be included within the set of memory object descriptions returned by the LIST request. The default is SVCDUMPRGN=YES. This keyword is ignored when V64SHARED=YES is specified.

,SVCDUMPRGN=YES
The memory objects with the SVCDUMPRGN=YES attribute are included in the set of memory objects returned.

,SVCDUMPRGN=NO
The memory objects with the SVCDUMPRGN=NO attribute are included in the set of memory objects returned.

,SVCDUMPRGN=ALL
All memory objects are included in the set of memory objects returned regardless if they have the SVCDUMPRGN=YES or SVCDUMPRGN=NO attributes.

,DUMP=ALL
,DUMP=LIKECSA
,DUMP=LIKESQA
,DUMP=LIKERGN
,DUMPPROTOCOL=NO
,ORDER=ASCENDING
,ORDER=DUMPPRIORITY
,DUMPPROTOCOL=YES
When V64SELECT=YES is specified, an optional parameter that specifies whether the memory object should be included within the set of memory object descriptions returned by the LIST request.

,DUMP=ALL
All memory objects, (regardless of what the SVCDUMP specification was on the IARV64 GETSTOR/GETCOMMON/SHAREMEMOBJ request) are included in the set of memory objects returned.

,DUMP=LIKECSA
The 64-bit common memory objects that have the DUMP=LIKECSA attribute specified or defaulted to on the IARV64 GETCOMMON request are included in the set of memory objects returned.

,DUMP=LIKESQA
The 64-bit common memory objects that have the DUMP=LIKESQA attribute specified or defaulted to on the IARV64 GETCOMMON request are included in the set of memory objects returned.

,DUMP=LIKERGN
The 64-bit private or 64-bit shared memory objects that have the DUMP=LIKERGN attribute specified or defaulted to on the IARV64 GETSTOR/SHAREMEMOBJ request are included in the set of memory objects returned.

,DUMPPROTOCOL=NO
,DUMPPROTOCOL=YES
An optional input parameter that specifies whether or not special selection criteria should be applied to the set of memory object descriptions returned by the LIST request. The DEFAULT is DUMPPROTOCOL=NO.

,DUMPPROTOCOL=NO
No additional selection criteria is applied.

,DUMPPROTOCOL=YES
When USERTOKEN=usertoken and SVCDUMPRGN=YES are specified, memory objects are returned according to certain selection criteria.

,ORDER=ASCENDING
,ORDER=DUMPPRIORITY
When V64SELECT=YES and DUMPPROTOCOL=NO is specified, an optional parameter that specifies the order in which the memory objects matching the selection criteria on the LIST request will be returned. The default is ORDER=ASCENDING.

,ORDER=ASCENDING
Memory objects that match the selection criteria are returned in ascending start address order.

,ORDER=DUMPPRIORITY
Memory objects that match the selection criteria are returned in dump priority order where memory objects with higher priority are listed before memory objects with lower priority. Within a dump priority level, memory objects will be listed based on ascending start address. ORDER=DUMPPRIORITY cannot be specified with V64SHARED=YES. ORDER=DUMPPRIORITY also can not be specified when SVCDUMPRGN=NO is specified.

,OWNERCOM=ALL
,OWNERCOM=HOME
,OWNERCOM=PRIMARY
,OWNERCOM=SYSTEM
,OWNERCOM=BYASID
An optional keyword input that specifies the owning entity of the 64-bit common memory objects to be included in the set returned.

,OWNERCOM=ALL
The 64-bit common memory objects belonging to all ASIDs are included in the set returned.

,OWNERCOM=HOME
The 64-bit common memory objects belonging to the HOME asid are included in the set returned.

,OWNERCOM=PRIMARY
The 64-bit common memory objects belonging to the PRIMARY asid are included in the set returned.

,OWNERCOM=SYSTEM
The 64-bit common memory objects belonging to the SYSTEM (not associated with an address space) are included in the set returned.

,OWNERCOM=BYASID
The 64-bit common memory objects belonging to a specific ASID are included in the set returned.

,OWNERASID=*ownerasid*
,OWNERASID=ALL
The name of an optional halfword integer input specifying the owning ASID of the 64-bit common memory objects to be included in the set returned.

,PAGEFRAMESIZE=ALL
,PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1MEG
An optional input parameter that specifies which memory objects should be included within the set of memory object descriptions returned by the LIST request. The DEFAULT is PAGEFRAMESIZE=ALL

,PAGEFRAMESIZE=ALL
All memory objects are included in the set of memory objects returned regardless of the page frame size.

,PAGEFRAMESIZE=4K
The memory objects which were backed by 4 KB frames are included in the set of memory objects returned.

,PAGEFRAMESIZE=1MEG
The memory objects which were backed by 1MEG frames are included in the set of memory objects returned.

,OWNERJOBNAME=*ownerjobname*
,OWNERJOBNAME=ALL
When V64SELECT=YES and V64COMMON=YES are specified, an optional input parameter specifying the owning jobname of the 64-bit common memory objects to be included in the set returned by the LIST request. The default is ALL.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,ATTRIBUTE=DEFS
,ATTRIBUTE=NOTOWNERGONE
,ATTRIBUTE=OWNERGONE
When V64SELECT=YES is specified, an optional parameter that specifies which memory objects should be included within the set of memory object descriptions returned by the LIST request. The default is ATTRIBUTE=DEFS.

,ATTRIBUTE=DEFS
The following ATTRIBUTE options are used to determine which memory objects should be included within the set of memory object descriptions returned by the LIST request: NOTOWNERGONE, OWNERGONE.

,ATTRIBUTE=NOTOWNERGONE
Include memory objects where the owner has not ended.

,ATTRIBUTE=OWNERGONE
Include memory objects where the owner has ended.

One or more values may be specified for the ATTRIBUTE parameter. If more than one value is specified, group the values within parentheses.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5g
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64

  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE
- **5**, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, 5

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=PAGEFIX option of IARV64

REQUEST=PAGEFIX allows you to fix physical pages within one or more nonshared memory objects. It makes virtual storage areas, above the bar, reside in central storage (also called real storage) and ineligible for page-out while the address space specified by the ALETVALUE is swapped into central storage.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Can be used only by callers running in supervisor state or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts. Enabled for I/O and external interrupts. Enabled or disabled for 64-bit common memory objects allocated with TYPE=DREF, or TYPE=PAGEABLE and the storage is in the first reference state.
Locks:	A local lock may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both above and below the bar.

Programming requirements

None.

Restrictions

Pages that are fixed must be unfixed before the task owning the memory object terminates. Otherwise the address space where the memory object resides is terminated.

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=PAGEFIX option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	

Syntax	Description
ƀ	One or more blanks must follow IARV64.
REQUEST=PAGEFIX	
,LONG=YES	**Default:** LONG=YES
,LONG=NO	
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4, 5	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

A required parameter. REQUEST=PAGEFIX specifies that the data within the specified ranges be pagefixed.

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PAGEFIX
A required parameter. REQUEST=PAGEFIX specifies that the data within the specified ranges be pagefixed.

PAGEFIX can only be requested for 64-bit private memory objects that are created using GETSTOR CONTROL=AUTH, and 64-bit common storage memory objects. PAGEFIX cannot be requested for 64-bit shared memory objects. 64-bit private or common storage memory objects backed by fixed 1 MB-page frames, or 64-bit private memory objects backed by fixed 2GB-page frames will be ignored.

PAGEFIX cannot be requested for a private memory object that was created using GETSTOR CONTROL=UNAUTH.

PAGEFIX cannot be requested for guard areas.

PAGEFIX specifies that the virtual storage areas are to reside in real storage and are ineligible for page-out while the address space is swapped in. This parameter does not prevent pages from being paged out when the entire address space is swapped out of real storage.

PAGEFIXed pages may be backed anywhere in real storage.

A page is considered PAGEFIXed until the number of valid PAGEUNFIXes issued for the page is equal to the number of valid PAGEFIXes previously issued for that page.

While a page is PAGEFIXed, the memory object, allocated with DETACHFIXED=NO, cannot be freed; if the system finds a PAGEFIXed area in the memory object, it abnormally ends the DETACH caller.

While a page is PAGEFIXed, the memory object allocated with DETACHFIXED=YES, can be freed successfully.

I/O can be done only to pages of memory objects that have been PAGEFIXed.

All I/O into virtual storage above the bar for an address space must be associated with the address space, that is, the ASID in the IOSB must be the ASID for the address space which owns the memory object. This is required so that I/O for the address space will be automatically purged during MEMTERM processing of the address space that owns the virtual storage above the bar or during I/O quiesce processing in preparation for swapping out the address space. The I/O must also be associated with the task which owns the memory object or one of its siblings. This is required so that all I/O is terminated and cleanup performed before the memory object is detached during task termination.

A resource manager must be provided to handle outstanding I/O when the task owning the memory object terminates. The resource manager must run before RSM's task termination resource manager and must ensure that all I/O into the virtual storage above the bar is complete and any fixed storage is unfixed. This is required for both normal and abnormal task termination. For example, this resource manager will be invoked through ABEND of the task termination if any virtual storage above the bar owned by the task is PAGEFIXED. This resource manager must ensure that all I/O into the memory object is complete. This is required for both normal and abnormal task termination.

PAGEFIX can be used only by callers running in supervisor state or with PSW key 0-7.

,LONG=YES
,LONG=NO
An optional input parameter that specifies whether the expected duration of the PAGEFIX is short or long. In general, a PAGEFIX is considered to be long if the time can be measured in seconds. The default is LONG=YES.

,LONG=YES
The PAGEFIX is expected to be of a long duration.

,LONG=NO
The PAGEFIX is expected to be of a short duration.

,RANGLIST=*ranglist*
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
Specifies the starting address of the data to be acted on.

The address specified must be within a memory object returned by GETSTOR CONTROL=AUTH or GETCOMMON.

The value must always be on a physical 4 KB-page boundary.The length of this field is 8 bytes.

NUMPAGES
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the storage is to be pagefixed. The ALETVALUE parameter is ignored for 64-bit common memory objects.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list. The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE
- **5**, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=PAGEIN option of IARV64

REQUEST=PAGEIN notifies the system that data within physical pages of one or more memory objects will be needed in the near future.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	A local lock may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=PAGEIN option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.

Syntax	Description
REQUEST=PAGEIN	
,RANGLIST=*ranglist*	*ranglist*: RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue*	*aletvalue*: RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,NUMRANGE=*numrange*	*numrange*: RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PAGEIN
A required parameter. REQUEST=PAGEIN notifies the system that data within the specified ranges is needed in the near future and should be retrieved from auxiliary storage, if possible. An attempt to PAGEIN a range which contains a guard area will cause an ABEND.

,RANGLIST=*ranglist*
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
denotes the starting virtual address of the data to be acted on.

The virtual address specified must be within an allocated memory object returned by GETSTOR, GETSHARED, or GETCOMMON.

The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.

NUMPAGES

Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the space in which the virtual storage is to be paged in.

The only supported values are 0 (primary address space) and 2 (home address space). The ALETVALUE parameter may be used only by callers executing in supervisor state or with a system (0-7) PSW key. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=PAGEOUT option of IARV64

REQUEST=PAGEOUT notifies the system that data within physical pages of one or more memory objects will not be used in the near future.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	A local lock may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=PAGEOUT option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.

Syntax	Description
REQUEST=PAGEOUT	
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PAGEOUT
A required parameter. REQUEST=PAGEOUT notifies the system that data within the specified ranges will not be used in the near future, i.e. for time measured in seconds (or longer), and are good candidates for paging.

Areas of the memory object that are PAGEFIXed or are in guard areas will not be affected.

,RANGLIST=*ranglist*
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
denotes the starting address of the data to be acted on.

The address specified must be within a memory object created by GETSTOR, GETSHARED, or GETCOMMON.

The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.

Note: If the starting address is backed by pageable 1 MB-page frames, use a value that is on a physical segment boundary. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

NUMPAGES
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

Note: If the range includes addresses backed by pageable 1 MB-page frames, specify a number of 4 KB-pages in multiples of 256. Failure to do so might result in the demotion of pageable 1 MB-page frames to pageable 4 KB-page frames.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the virtual storage is to be paged out.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=PAGEUNFIX option of IARV64

Use REQUEST=PAGEUNFIX to unfix physical pages within one or more non-shared or common memory objects.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Can be used only by callers running in supervisor state or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	A local lock may be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=PAGEUNFIX option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.

Syntax	Description
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.
REQUEST=PAGEUNFIX	
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,COND=NO	**Default:** COND=NO
,COND=YES	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3, 4, 5	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PAGEUNFIX
A required parameter. REQUEST=PAGEUNFIX specifies that a range of storage has no I/O in progress and will no longer be used for I/O or will no longer be referenced disabled.

PAGEUNFIX can only be requested for 64-bit private memory objects that are created using GETSTOR CONTROL=AUTH, and 64-bit common storage memory objects. PAGEUNFIX cannot be requested for 64-bit shared memory objects. 64-bit private or common storage memory objects backed by fixed 1 MB-page frames, or 64-bit private memory objects backed by fixed 2GB-page frames will be ignored.

A page is considered PAGEFIXed until the number of valid PAGEUNFIXes issued for the page is equal to the number of valid PAGEFIXes previously issued for that page.

If a PAGEUNFIX is issued for a page that is not PAGEFIXed, the caller will be abnormally ended.

The PAGEUNFIX keyword can be used only by callers running in supervisor state or with PSW key 0-7.

,RANGLIST=*ranglist*
A required input parameter, of a range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
Denotes the starting address of the data to be acted on.

The address specified must be within a memory object returned by GETSTOR CONTROL=AUTH.

The value must always be on a physical 4 KB-page boundary.

The length of this field is 8 bytes.

NUMPAGES
Contains the number of physical 4 KB-pages in the area.

The number of 4 KB-pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the storage is to be unfixed.

The only supported values are 0 (primary) and 2 (home). ALETVALUE can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the range list. The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,COND=NO
,COND=YES
This is an optional input parameter that specifies whether the request is unconditional or conditional. The DEFAULT value is COND=NO.

COND=NO
The request is unconditional. The request is abnormally ended when the request cannot be satisfied.

COND=YES
The request is conditional. The request is not abnormally ended for unfixed pages in the range specified.

Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. If you code COND=YES and there are unfixed pages in the range specified, instead of the request being abnormally ended, the request will complete but a return code will be set to indicate that the request was completed abnormally. In this case, the unfixed pages skipped and all the fixed pages will be unfixed.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA

  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE
- **5**, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=PROTECT option of IARV64

REQUEST=PROTECT requests that data within one or more memory objects be made read-only.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15. The caller must have a PSW key of 0 or a PSW key that matches the storage to be protected. The caller must be running in supervisor state or with PSW key 0-7 to use the ALETVALUE keyword.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts for 64-bit private. Enabled or disabled for I/O and external interrupts for 64-bit common storage.
Locks:	You may hold the local lock for the target address space. If you hold the local lock, you can also hold the CMS lock. For disabled callers, no spin locks higher than the RSM locks can be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=PROTECT option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.
REQUEST=PROTECT	
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).

Syntax	Description
,NUMRANGE=1	**Default:** NUMRANGE=1
,AMOUNTSIZE=1MEG	**Default:** AMOUNTSIZE=4K. AMOUNTSIZE=1MEG is the default when the data to be acted on is from a fixed 1 MB-page frames request. AMOUNTSIZE cannot be specified when the data to be acted on is from a fixed 2 GB-page frames request.
,AMOUNTSIZE=4K	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=0, 1, 2 , 3, 4, 5	**Default:** PLISTVER=IMPLIED_VERSION
,MF=S ,MF=(L, *list addr*) ,MF=(L, list addr, attr) ,MF=(L ,list addr, 0D) ,MF=(E, list addr) ,MF=(E ,list addr, COMPLETE) ,MF=(M, list addr) ,MF=(M ,list addr, COMPLETE) ,MF=(M, list addr, NOCHECK)	Default: MF=S *list addr:* RS-type address or register (1) - (12).

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=PROTECT
A required parameter. REQUEST=PROTECT specifies that a range of virtual storage be made read-only.

Areas of the memory object that are in guard areas or hidden will not be affected.

,RANGLIST=*ranglist*
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
Denotes the starting address of the data to be acted on.

The address specified must be within a created memory object returned by GETSTOR or GETCOMMON.

The value must be on a segment (1M) boundary when either:

- PAGEFRAMESIZE=1MEG|1M was specified on the IARV64 request (GETSTOR or GETCOMMON) that created the memory object, or
- PAGEFRAMESIZE=MAX was specified on the IARV64 request (GETSTOR or GETCOMMON) that created the memory object and the address specified is backed by fixed 1M page frames.

The value must be on a region (2G) boundary when PAGEFRAMESIZE=2G is specified. Otherwise, the value must be on a page (4K) boundary.

Note: If the range includes addresses backed by pageable 1 MB-page frame, specify a value on a physical segment boundary. Otherwise, demotion of pageable 1 MB-page frame to pageable 4 KB-page frames might occur.

The length of this field is 8 bytes.

AMOUNT
Contains the number of 4K, 1M, or 2G pages to be acted on.

The number of 4K, 1M, or 2G pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

The amount contains the number of 1M pages to be acted on when the specified VSA is backed by fixed large page frames (1M).

The amount contains the number of 2G pages to be acted on when the specified VSA is backed by fixed 2G page frames (2G).

Note: If the range includes addresses backed by pageable 1M page frames, specify a number of 4K pages in multiples of 256; otherwise, demotion of pageable 1M page frames to pageable 4K page frames might occur.

,AMOUNTSIZE=4K
,AMOUNTSIZE=1MEG
An optional input parameter that specifies how the AMOUNT value specified on the RANGLIST parameter is treated. The default value is AMOUNTSIZE=4K.

When the data to be acted on is from a fixed 1M page frames request (TYPE=FIXED and PAGEFRAMESIZE=1M is specified, or TYPE is not specified and PAGEFRAMESIZE=1MEG is specified, or PAGEFRAMESIZE=MAX is specified and the address specified is backed by fixed 1M page frames), the default is AMOUNTSIZE=1MEG. Only AMOUNTSIZE=1MEG can be specified.

When the data to be acted on is from a fixed 2G page frames request (TYPE=FIXED and PAGEFRAMESIZE=2G is specified), AMOUNTSIZE cannot be specified. The AMOUNT value specified on the RANGLIST parameter is treated as the number of 2G pages to be acted on.

4K
The AMOUNT value specified on the RANGLIST parameter is the number of 4K page frames to be acted on.

1MEG
The AMOUNT value specified on the RANGLIST parameter is the number of 1M page frames to be acted on.

,ALETVALUE=*aletvalue*
,ALETVALUE=*0*
An optional input parameter that indicates the ALET of the address space in which the virtual storage is to be protected.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=*numrange*
,NUMRANGE=*1*
An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16.

Default: 1

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - GETSHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM

- TYPE
- UNLOCKED
- USERTOKEN
- V64COMMON

- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE
- **5**, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

,list addr

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK
This parameter specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=SHAREMEMOBJ option of IARV64

REQUEST=SHAREMEMOBJ requests that the address space be given access to one or more specified shared memory objects.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be running in supervisor state or with PSW key 0-7.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	When a local lock is held for a request the lock must be for the address space specified (or defaulted) by the input ALETVALUE.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None.

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=SHAREMEMOBJ option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IARV64.
IARV64	
␣	One or more blanks must follow IARV64.
REQUEST=SHAREMEMOBJ	
,USERTKN=*usertkn*	*usertkn:* RS-type address or address in register (2) - (12).
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).

Syntax	Description
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,SVCDUMPRGN=YES	**Default:** SVCDUMPRGN=YES
,DUMPPRIORITY=*dumppriority*	
,SVCDUMPRGN=NO	
,COND=NO	**Default**: COND=NO
,COND=YES	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0, 1, 2, 3 or 4	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IARV64 macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=SHAREMEMOBJ
REQUEST=SHAREMEMOBJ requests that the caller wants to be given shared access to the specified memory object. The memory object specified must be a SHARED memory object, such as the result of a GETSHARED invocation.

,USERTKN=*usertkn*
A required doubleword input parameter that identifies the user token to be associated with the memory object. This can be used on a later DETACH request to free all memory objects associated with this value.

To avoid inadvertent collisions in the values specified, the left word (bits 0-31) should represent an address of some storage related to the caller. The system enforces the rule that the left word is non-zero for authorized callers. The right word should represent the virtual address of some storage related to the caller, which could be a control block address, an entry point address, etc.; the choice of which to use is made by the application.

Note: The scope of a user token value is the entire z/OS image. It can be associated with shared, common, or private memory objects.

To code: Specify the RS-type address, or address in register (2)-(12), of a required doubleword field that identifies the user token to be associated with the shared memory object.

,RANGLIST=*ranglist*
A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA
VSA denotes the starting address of the data to be acted on. The virtual address must be within a memory object returned by GETSHARED (not GETSTOR or GETCOMMON).

The length of this field is 8 bytes.

RESERVED
Reserved for future use, must be in binary zeros.

The length of this field is 8 bytes.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list. The value specified must be no greater than 16. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space that will be given access to the shared memory object.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,SVCDUMPRGN=YES
,SVCDUMPRGN=NO
An optional parameter that specifies whether the memory object should be included in an SVC dump when region is requested. The default is SVCDUMPRGN=YES.

,SVCDUMPRGN=YES
The memory object should be included in an SVC dump when RGN is specified on SDATA.

,DUMPPRIORITY=*dumppriority*
The name of an optional one-byte integer input parameter that specifies the dump priority of the memory object. This must be a non-zero value in the range of 1-99, with 1 being the highest priority and 99 being the lowest. The default value is 99.

,SVCDUMPRGN=NO
The memory object should not be included in an SVC dump when RGN is specified on SDATA.

,COND=NO
,COND=YES
An optional keyword input that specifies whether the request is unconditional or conditional. The default is COND=NO.

,COND=NO
The request is unconditional. The request will be abnormally ended when the request cannot be satisfied.

,COND=YES
The request is conditional. The request will not be abnormally ended for resource unavailability.

Use of COND=YES does not prevent all abnormal terminations. For instance, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3 or 4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - AMOUNTSIZE
  - DETACHFIXED

  - DOAUTHCHECKS
  - DUMP
  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3 or 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

REQUEST=UNPROTECT option of IARV64

REQUEST=UNPROTECT requests that data within one or more memory objects be made modifiable.

The IARV64 REQUEST=UNPROTECT unprotects pages/segments within 64-Bit Private or 64-Bit Common memory objects. 64-bit fixed 2 GB private memory objects cannot be protected.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15. The caller must have a PSW key of 0 or a PSW key that matches the storage to be unprotected. The caller must be running in supervisor state or with PSW key 0-7 to use the ALETVALUE keyword.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts for 64-bit private. Enabled or disabled for I/O and external interrupts for 64-bit common storage.
Locks:	You may hold the local lock for the target address space. If you hold the local lock, you may also hold the CMS lock. For disabled callers no spin locks higher than the RSM locks can be held.
Control parameters:	Control parameters must be in the primary address space and can reside both below and above the bar.

Programming requirements

None

Restrictions

This macro supports multiple versions. Some keywords are unique to certain versions. See PLISTVER parameter description.

Input register information

Before issuing the IARV64 macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None

Syntax

The REQUEST=UNPROTECT option of the IARV64 macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	One or more blanks must follow IARV64.
REQUEST=UNPROTECT	

Syntax	Description
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue*	*aletvalue:* RS-type address or address in register (2) - (12).
,ALETVALUE=0	**Default:** ALETVALUE=0
,NUMRANGE=*numrange*	*numrange:* RS-type address or address in register (2) - (12).
,NUMRANGE=1	**Default:** NUMRANGE=1
,AMOUNTSIZE=1MEG	**Default:** AMOUNTSIZE=4K. AMOUNTSIZE=1MEG is the default when the data to be acted on is from a fixed 1 MB-page frames request. AMOUNTSIZE cannot be specified when the data to be acted on is from a fixed 2 GB-page frames request.
,AMOUNTSIZE=4K	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=0, 1, 2, 3, 4, 5	**Default:** PLISTVER=IMPLIED_VERSION
,MF=S ,MF=(L, *list addr*) ,MF=(L, list addr, attr) ,MF=(L, list addr, 0D) ,MF=(E, list addr) ,MF=(E, list addr, COMPLETE) ,MF=(M, list addr) ,MF=(M ,list addr, COMPLETE) ,MF=(M, list addr, NOCHECK)	Default: MF=S *list addr:* RS-type address or register (1) - (12).

Parameters

The parameters are explained as follows:

name
A required parameter. REQUEST=UNPROTECT specifies that a range of virtual storage be made modifiable.

REQUEST=UNPROTECT
A required parameter. REQUEST=UNPROTECT specifies that a range of virtual storage be made modifiable.

Areas of the memory object that are in guard areas or hidden will not be affected.

,RANGLIST=*ranglist*

A required input parameter. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 16 bytes long. A description of the fields in each entry follows:

VSA

Denotes the starting address of the data to be acted on.

The address specified must be within a created memory object returned by GETSTOR or GETCOMMON.

The value must be on a segment (1M) boundary when either:

- PAGEFRAMESIZE=1MEG|1M is specified, or
- PAGEFRAMESIZE=MAX is specified and the address specified is backed by fixed 1M page frames.

The value must be on a region (2G) boundary when PAGEFRAMESIZE=2G is specified. Otherwise, the value must be on a page (4K) boundary.

Note: If the range includes addresses backed by pageable 1 MB-page frames, specify a value on a physical segment boundary. Otherwise, demotion of pageable 1 MB-page frames to pageable 4 KB-page frames could occur.

The length of this field is 8 bytes.

AMOUNT

Contains the number of 4K, 1M, or 2G pages to be acted on.

The number of 4K, 1M, or 2G pages specified starting with the specified VSA must lie within a single memory object.

The length of this field is 8 bytes.

The amount contains the number of 1M pages to be acted on when the specified VSA is backed by fixed large page frames (1M).

The amount contains the number of 2G pages to be acted on when the specified VSA is backed by fixed 2G page frames (2G).

Note: If the range includes addresses backed by pageable 1M page frames, specify a number of 4K pages in multiples of 256; otherwise, demotion of pageable 1M page frames to pageable 4K page frames could occur.

,AMOUNTSIZE=4K
,AMOUNTSIZE=1MEG

An optional input parameter that specifies how the AMOUNT value specified on the RANGLIST parameter is treated. The default value is AMOUNTSIZE=4K.

When the data to be acted on is from a fixed 1M page frames request (TYPE=FIXED and PAGEFRAMESIZE=1M is specified, or TYPE is not specified and PAGEFRAMESIZE=1MEG is specified, or PAGEFRAMESIZE=MAX is specified and the address specified is backed by fixed 1M page frames), the default is AMOUNTSIZE=1MEG. Only AMOUNTSIZE=1MEG can be specified.

When the data to be acted on is from a fixed 2G page frames request (TYPE=FIXED and PAGEFRAMESIZE=2G is specified), AMOUNTSIZE cannot be specified. The AMOUNT value specified on the RANGLIST parameter is treated as the number of 2G pages to be acted on.

4K

The AMOUNT value specified on the RANGLIST parameter is the number of 4K page frames to be acted on.

1MEG

The AMOUNT value specified on the RANGLIST parameter is the number of 1M page frames to be acted on.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the address space in which the virtual storage is to be unprotected.

The only supported values are 0 (primary) and 2 (home). The ALETVALUE parameter can be used only by callers running in supervisor state or with PSW key 0-7. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,NUMRANGE=*numrange*
,NUMRANGE=1
An optional input parameter that specifies the number of entries in the supplied range list.

The value specified must be no greater than 16. The default is 1.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0, 1, 2, 3, 4, 5
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, you can always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - CONVERTSIZE64
  - CONVERTSTART
  - GUARDSIZE64
  - V64SHARED
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - GETCOMMON
  - PAGEPROTECT
  - PAGEUNPROTECT
  - AMOUNTSIZE
  - DETACHFIXED
  - DOAUTHCHECKS
  - DUMP

  - DUMPPRIORITY
  - DUMPPROTOCOL
  - LOCALSYSAREA
  - MEMLIMIT
  - OPTIONVALUE
  - ORDER
  - OWNERASID
  - OWNERCOM
  - TYPE
  - UNLOCKED
  - USERTOKEN
  - V64COMMON
- **3**, supports both the following parameters and parameters from versions 0, 1, 2:
  - ATTRIBUTE
  - OWNERJOBNAME
  - TRACKINFO
- **4**, supports both the following parameter and parameters from versions 0, 1, 2, 3:
  - DMAPAGETABLE
- **5**, supports both the following parameters and parameters from versions 0, 1, 2, 3, 4:
  - UNITS
  - UNITSIZE

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, 4, or 5

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

You can use the modify and execute forms in the following order:

- Use MF=(M,list_addr,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,list_addr,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,list_addr,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK
This parameter specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes

Regardless of specifying COND=NO or COND=YES, IARV64 might abnormally terminate with hexadecimal abend code DC2 and a 4-byte unique reason code. See the description of the COND parameter for each IARV64 request type being used, and see DC2 in *z/OS MVS System Codes* for an explanation of the abend and programmer response.

For specific abend examples and the causes, see Proper serialization of shared memory objects in *z/OS MVS Programming: Extended Addressability Guide*.

Return and reason codes

When the IARV64 macro returns control to your program GPR 15 (and *retcode*, when you code RETCODE) contains a return code. When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

If you are coding a conditional request (COND=YES), check the results of the request by validating the return and reason codes. Note that COND=YES does not necessarily prevent abends.

The following table identifies the hexadecimal return and reason codes. IBM support personnel may request the entire reason code, including the **xx** value.

Table 42. Return and Reason Codes for the IARV64 Macro

Return Code	Reason Code	Meaning and Action
00	—	**Meaning**: Successful completion **Action**: None required
02	—	**Meaning**: Successful completion, with exception. For a LIST request, IARV64 requests have been issued since the previous call to LIST. **Action**: Reissue the call if you need the information pertaining to those recent IARV64 requests.

Table 42. Return and Reason Codes for the IARV64 Macro (continued)

Return Code	Reason Code	Meaning and Action
04	—	**Meaning**: Successful completion, with exception. For a LIST request, there are additional memory objects which were not returned on this call to LIST. For a CHANGEGUARD request, one or more segments in the memory object are already in the requested state. For a DETACH request, the memory object task token does not match the TToken of the caller. For GETSTOR, GETCOMMON request with PAGEFRAMESIZE(MAX), no large frame was available, PAGEFRAMESIZE(4K) was used. **Action**: For a CHANGEGUARD request, if this is unexpected, then ensure that the correct values for CONVERTSIZE, CONVERTSIZE64, or CONVERTSTART are specified. If it is already known that some segments may be in the requested state, then no action is required. For a LIST request, issue the LIST call again to get the additional information. For GETSTOR, GETCOMMON request with PAGEFRAMESIZE(MAX), no action is required, the memory object was backed by a 4K page.
06	—	**Meaning**: Successful completion, with exception. For a LIST request, there are additional memory objects which were not returned on this call to LIST and IARV64 requests have been issued since the previous call to LIST. For a COUNTPAGES request, the counts are incomplete because additional IARV64 requests have been issued during COUNTPAGES processing. **Action**: Issue the LIST or COUNTPAGES call again to get the additional information.

Table 42. Return and Reason Codes for the IARV64 Macro (continued)		
Return Code	**Reason Code**	**Meaning and Action**
08	—	**Meaning**: The request is rejected because of non-system failure. This reason code could be issued for a conditional IARV64 request. In this case this reason code is the same as the DC2 reason code issued from an unconditional IARV64 request. See DC2 in *z/OS MVS System Codes* for an explanation and programmer response. Otherwise, if it is not there, then it has one of the following meanings: For a GETSTOR or CHANGEGUARD request, there was insufficient storage to satisfy the request. For a DETACH request, there were no memory objects deleted because none matched the user token provided. For a LIST request, there were no memory objects returned because no memory objects match the selection criteria. **Action**: For a DETACH request, make sure that the user token was correct. For a LIST request, no action is required. For other requests, see DC2 in *z/OS MVS System Codes* for an explanation and programmer response.
0C	—	**Meaning**: The request is rejected because of system failure. This reason code could be issued for a conditional IARV64 request. In this case this reason code is the same as the DC2 reason code issued from an unconditional IARV64 request. See DC2 in *z/OS MVS System Codes* for an explanation and programmer response. Otherwise, if it is not there, then it has the following meaning: • For a GETSTOR request, there was insufficient storage to build the control structure. • For a COUNTPAGES request, there was an error during UNLOCKED=YES processing that indicates the page table structure has changed. **Action**: • For a GETSTOR request, free storage within address space so control structures can be built. • For a COUNTPAGES request, try the request again. • For all other requests, see DC2 in *z/OS MVS System Codes* for an explanation and programmer response.

Example

Operation:

1. Get 2 MB above the bar
2. Page-fix the first 1 MB of that storage
3. Page-unfix that first 1 MB

4. Free the storage

The code is as follows:

```
         SYSSTATE AMODE64=YES
************************************************************
* Get storage above 2G                                     *
************************************************************
         IARV64 REQUEST=GETSTOR,SEGMENTS=NUMSEG,             *
               ORIGIN=OUTORG,RETCODE=LRETCODE,               *
               RSNCODE=LRSNCODE,CONTROL=AUTH,                *
               MF=(E,V64L)
*
* Place code to check return/reason codes here
*
*
* Build the Range List for Pagefix
*
         LG    1,OUTORG
         STG   1,RLSTART
         LG    1,ONEMEG         Number of pages in 1-meg
         STG   1,RLEND
         LA    1,RL
         LLGTR 1,1
         STG   1,RLADDR

************************************************************
* Page-fix that storage                                    *
* Defaults to NUMRANGE=1.                                  *
************************************************************
*
         IARV64 REQUEST=PAGEFIX,RANGLIST=RLADDR,             *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,V64L)
*
* Place code to check return/reason codes here
*
************************************************************
* Page-unfix that storage                                  *
* Defaults to NUMRANGE=1.                                  *
************************************************************
*
         IARV64 REQUEST=PAGEUNFIX,RANGLIST=RLADDR,           *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,V64L)
*
* Place code to check return/reason codes here
*
************************************************************
* Free the storage                                         *
************************************************************
         IARV64 REQUEST=DETACH,MEMOBJSTART=OUTORG            *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,V64L)
*
* Place code to check return/reason codes here
*
NUMSEG   DC    AD(2)
ONEMEG   DC    AD((1024*1024)/4096)  Num of pages in a megabyte
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
OUTORG   DS    AD
RLADDR   DS    0D                      Start of 16-byte range list
RLSTART  DS    AD                    Address of memory object
RLEND    DS    AD                    Number of pages
         IARV64 MF=(L,V64L)
```

Chapter 35. IAZXCTKN — Client token compare service

Description

Use the IAZXCTKN macro to compare two client tokens. This service should be used anytime client tokens have to be compared.

Environment

Environmental factor	Requirement
Minimum authorization:	Problem state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary Secondary Access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	None

Programming requirements

Include the IAZXCTKN mapping macro.

Restrictions

None.

Input register information

Before issuing the IAZXCTKN macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Destroyed

2-13
Unchanged

14
Destroyed

15
Return code

Performance implications

None.

Syntax

The IAZXCTKN macro is written as follows:

Syntax	Description
CTOKENA=*ctoken addr*	*ctoken addr*: RX-type address or address in register (2) - (13).
CTOKENB=*ctoken addr*	*ctoken addr*: RX-type address or address in register (2) - (13).

Parameters

The IAZXCTKN parameters are explained as follows:

,CTOKENA=*ctoken addr*
Specifies the address of an 80-byte area containing one CTOKEN to be compared.

,CTOKENB=*ctoken addr*
Specifies the address of an 80-byte area containing one CTOKEN to which CTOKENA is to be compared.

ABEND codes

None.

Return codes

When IAZXCTKN macro returns control to your program. GPR 15 contains a return code.

Table 43. Return Codes for the IAZXCTKN Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: CTOKENA and CTOKENB contain the same significant information.
04	**Meaning**: CTOKENA and CTOKENB contain different significant information and CTOKENA's sort information is less than CTOKENB's sort information.
08	**Meaning**: CTOKENA and CTOKENB contain different significant information and CTOKENB's sort information is less than CTOKENA's sort information.
12	**Meaning**: CTOKENA and CTOKENB contain different significant information and at least one of these ctokens contains no sort information.
16	**Meaning**: CTOKENA and CTOKENB contain different significant information but the sort information in the two ctokens is equal. This indicates a "collision" of the two ctokens.

Example

```
IAZXCTKN CTOKENA=BILL,CTOKENB=SAM
```

Chapter 36. IAZXJSAB — Obtain information about a currently running job

Description

Use the CREATE function with TYPE=SUBTASK to set or request job scheduler information for the current task. When an application is doing work on behalf of another work unit, the CREATE or UPDATE request identifies the work unit for whom the work is being done. IAZXJSAB provides the following services:

- Create a JSAB.
- Read from a JSAB.
- Update a JSAB.
- Delete a JSAB.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	CREATE, DELETE and UPDATE: APF authorization, supervisor state or system key READ: Problem or supervisor state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	If the caller specifies the ASCB parameter, any PASN, any HASN, any SASN; otherwise, PASN=HASN is required.
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	None

Programming requirements

Include these mapping macros in your program: IAZJSAB, IHAASCB and IHAASSB. If you do not code the ASCB parameter, also include these additional mapping macros: IHAPSA, IKJTCB, and IHASTCB.

If you need more information about:

Mapping macro:	Look in:	Under the name:
IAZJSAB	*z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)	JSAB
IHAASCB	*z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)	ASCB
IHAASSB	*z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)	ASSB
IHAPSA	*z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)	PSA

Mapping macro:	Look in:	Under the name:
IHASTCB	*z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)	STCB
IKJTCB	*z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)	TCB

Restrictions

The following restrictions apply:

- You must not create an address space JSAB.
- You may only update or delete a JSAB that you have created.

Input register information

Before issuing the IAZXJSAB macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IAZXJSAB macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IAZXJSAB macro.
IAZXJSAB	
ƀ	One or more blanks must follow IAZXJSAB macro.
CREATE READ UPDATE DELETE	
,TYPE=SUBTASK	
	Default: ADDRSP (Restricted use)
,ASCB=*ascb addr*	*ascb addr*: RX-type address or register (2) - (12).
	Default: The address of the ASCB for the caller's home address space.
,COMPID=*compid addr*	*compid addr*: RS-type address or register (2) - (12).
,CODELEV=*codelev addr*	*codelev addr*: RX-type address or register (2) - (12).
,WORKID=*workid addr*	*workid addr*: RS-type address or register (2) - (12).
,JOBID=*jobid addr*	*jobid addr*: RS-type address or register (2) - (12).
,JOBNAME=*jobname addr*	*jobname addr*: RS-type address or register (2) - (12).
,PREFIX=*prefix addr*	*prefix addr*: RS-type address or register (2) - (12).
,USERID=*userid addr*	*userid addr*: RS-type address or register (2) - (12).
,EXECST=*execst addr*	*execst addr*: RS-type address or register (2) - (12).

Syntax	Description
,XCFGPNM=*xcfgpnm addr*	*xcfgpnm addr*: RS-type address or register (2) - (12).
,JESTAT=*jestat addr*	*jestat addr*: RS-type address or register (2) - (12).
,JSABLVL=*jsablvl addr*	*jsablvl addr*: RS-type address or register (2) - (12).
,JOBCORR=*jobcorr addr*	*jobcorr addr*: RS-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

CREATE | READ | UPDATE | DELETE
Requests the type of IAZXJSAB function.

,TYPE=SUBTASK
Specifies the type of JSAB to be created or deleted. TYPE is valid only for the CREATE and DELETE services. You must code TYPE=SUBTASK, as the default is TYPE=ADDRSP.

,ASCB=*ascb addr*
Specifies the address of an address space control block (ASCB).

ASCB is valid only for the READ and DELETE services. The default value is the address of the ASCB that represents the caller's home address space.

,COMPID=*compid addr*
With the CREATE service, COMPID specifies the location of the 4-character name of the subsystem that is creating the JSAB. With the READ service, COMPID specifies the location where the system is to return the 4-character name of the subsystem that created the JSAB. If JES2 or JES3 created the JSAB, the identifier is the common name of the JES (such as JES2 or JES3) and not the name of the JES2 or JES3 address space. If APPC/MVS created the JSAB, the identifier is ASCH.

COMPID is required for the CREATE service and is optional for the READ service. COMPID is not valid on the UPDATE and DELETE services.

,CODELEV=*codelev addr*
Specifies the code level of the creating component. Valid values are 0-255.

CODELEV is required for the CREATE service and is optional for the READ service. CODELEV is not valid on the UPDATE and DELETE services.

,WORKID=*workid addr*
Specifies the location where the system is to return the 8-character work unit identifier. The system returns identical information for the work unit ID and job ID.

WORKID is not valid on the DELETE service and is optional on the CREATE, READ, and UPDATE services.

,JOBID=*jobid addr*
Specifies the location where the system is to return the 8-character job identifier. The system returns identical information for the work unit ID and job ID.

JOBID is not valid on the DELETE service and is optional on the CREATE, READ, and UPDATE services.

,JOBNAME=*jobname addr*
Specifies the location where the system is to return the 8-character job name.

JOBNAME is not valid on the DELETE service and is optional on the CREATE, READ, and UPDATE services.

,PREFIX=*prefix addr*
Specifies the location where the system is to return the 8-character message prefix. In a JES2 system, the prefix is the job ID. In a JES3 system, the prefix is the job name.

PREFIX is not valid on the DELETE service and is optional on the CREATE, READ, and UPDATE services.

,USERID=*userid addr*
Specifies the location where the system is to return the 8-character user ID.

USERID is not valid on the DELETE service and is optional on the CREATE, READ, and UPDATE services.

,EXECST=*execst addr*
Specifies the location where the system is to return the 8-byte execution start time, in time-of-day (TOD) clock format.

EXECST is not valid on the DELETE service.

,XCFGPNM=*xcfgpnm addr*
Specifies the location where the system is to return the 8-character XCF group name of the subsystem that created the JSAB. The XCF group name is available only if JES2 created the JSAB.

XCFGPNM is not valid on the DELETE service and is optional on the CREATE, READ, and UPDATE services.

,JESTAT=*jestat addr*
Specifies the location where the system is to return the 8-byte JES status for the address space.

For the meaning of values that can be returned to the specified address, see the field JSABJSTA in the mapping macro IAZJSAB in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

JES status only applies on the job level. Therefore, using the JESTAT keyword forces the access to the address space level JSAB. JESTAT is not compatible with TYPE(SUBTASK). For READ and UPDATE, JESTAT can only be used with the JOBCORR keyword—JESTAT cannot be used together with any other job attribute.

,JSABLVL=*jsablvl addr*
Specifies the location where the system is to return the 1-byte level of the JSAB to be used for the READ request. If the value returned is 4, it indicates a subtask level JSAB.

JSABLVL is valid only on the READ service.

,JOBCORR=*jobcorr addr*
Specifies the location where the system is to return the 64-character job correlator field. Because JOBCORR only applies on the job level, using the JOBCORR keyword forces the access to the address space level JSAB. JOBCORR is not compatible with TYPE(SUBTASK). For READ and UPDATE, JOBCORR can only be used with the JESTAT keyword—JOBCORR cannot be used with any other job attribute.

JOBCORR is optional for CREATE, READ and UPDATE. JOBCORR is not valid for DELETE.

ABEND codes

None.

Return codes

When IAZXJSAB macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes.

Table 44. Return and Reason Codes for the IAZXJSAB Macro

Return Code	Meaning and Action
0	**Meaning**: Processing completed successfully. **Action**: None.
4	**Meaning**: Storage was not obtained or released for the JSAB. (CREATE and DELETE only.) **Action**: None.
8	**Meaning**: The JSAB was not found. No information was returned. **Action**: None required; however, you might want to make sure the specified ASCB address is correct.
12	**Meaning**: The requested field does not exist in the active JSAB. **Action**: None.

Example

Obtain the job ID of the current address space.

```
         IAZXJSAB READ,JOBID=MYJOBID
          .
          .
          .
MYJOBID  DS    CL8
```

Chapter 37. IEAARR — Establish an associated recovery routine (ARR)

Description

IEAARR allows you to request that the system establish an associated recovery routine (ARR) while calling a target routine. In this case, the system performs the stacking PC instruction, then give control to your routine (the target routine). When the target routine returns control, the system issues the corresponding PR instruction. Asynchronous exist processing will be prohibited while the ARR is running.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller is not required to hold any locks on entry. The caller may hold the local, CMS, or CPU lock.
Control parameters:	None.

Programming requirements

The caller must include the IHAECVT mapping macro.

Restrictions

IEAARR must not be issued while a functional recovery routine (FRR) is established.

TARGETSTATE=PROB should only be issued by a caller currently running in problem state. TARGETSTATE=SUP should only be issued by a caller currently running in supervisor state.

Input register information

Before issuing the IEAARR macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
The value placed in register 0 by the target routine prior to its returning to the system.

1
The value placed in register 1 by the target routine prior to its returning to the system.

2-13
Unchanged

14
Used as a work register by the system

15
The value placed in register 15 by the target routine prior to its returning to the system.

When control returns to the caller, the ARs contain:

Register
Contents

0
The value placed in access register 0 by the target routine prior to its returning to the system.

1
The value placed in access register 1 by the target routine prior to its returning to the system.

2-13
Unchanged

14
Used as a work register by the system

15
The value placed in access register 15 by the target routine prior to its returning to the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IEAARR macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEAARR.
IEAARR	
␣	One or more blanks must follow IEAARR.
ARRPTR=*arrptr*	*arrptr:* RX-type address or address in register (2) - (12).
ARR=*arr*	*arr:* RX-type address or address in register (2) - (12).
,DYNSTORAGE=AVAIL	**Default:** DYNSTORAGE=AVAIL

Syntax	Description
,DYNSTORAGE=NOTAVAIL	
,ARRPARAMPTR=*arrparamptr*	*arrparamptr:* RX-type address or address in register (2) - (12).
,ARRPARAMPTR64=*arrparamptr64*	*arrparamptr64:* RX-type address or address in register (2)-(12), of a 64-bit pointer field.
,ARRPARAM=*arrparamp*	*arrparamp:* RX-type address or address in register (2) - (12).
,ARRPARAM64=*arrparam64*	*arrparam64:* RX-type address or address in register (2) - (12).
,PARAMPTR=*paramptr*	*paramptr:* RX-type address or address in register (2) - (12).
,PARAMPTR64=*paramptr64*	*paramptr64:* RX-type address or address in register (2)-(12), of a 64-bit pointer field.
,PARAM=*param*	*param:* RX-type address or address in register (2) - (12).
,PARAM64=*param64*	*param64:* RX-type address or address in register (2) - (12).
,TARGETPTR=*targetptr*	*targetptr:* RX-type address or address in register (2) - (12).
,TARGET=*target*	*target:* RX-type address or address in register (2) - (12).
,TARGETSTATE=PROB	
,TARGETSTATE=SUP	
,ASYNCH=NO	**Default:** ASYNCH=NO
,ASYNCH=YES	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEAARR macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

DYNSTORAGE=AVAIL
DYNSTORAGE=NOTAVAIL
An optional parameter that indicates whether this routine is sensitive to your having dynamic storage. The default is DYNSTORAGE=AVAIL.

DYNSTORAGE=AVAIL
Indicates that you have dynamic storage available.

DYNSTORAGE=NOTAVAIL
Indicates that you do not have dynamic storage available. The parameters are defined so that you can place each parameter value in a register and in so doing avoid the need to place parameter values into dynamic storage.

ARRPTR=*arrptr*
When DYNSTORAGE=AVAIL is in effect, a required input parameter that contains the address of the associated recovery routine. This routine gets control from RTM according to normal z/OS recovery protocols. As it is an ARR, it will get control in AMODE 31.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.

ARR=*arr*
When DYNSTORAGE=NOTAVAIL is specified, a required input parameter that is the associated recovery routine. This routine gets control from RTM according to normal z/OS recovery protocols. As it is an ARR, it will get control in AMODE 31.

To code: Specify the RX-type address, or address in register (2)-(12), of the associated recovery routine.

,ARRPARAMPTR=*arrparamptr*
When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is not in effect, a required input parameter that contains the address of the parameter area that is to be passed to the ARR upon error. The address is placed in the first four bytes of the area pointed to by SDWAPARM and in GPR 2. Note that the second four bytes of the area pointed to by SDWAPARM will not contain interface information.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.

,ARRPARAM=*arrparam*
When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is not in effect, a required input parameter that is the parameter area that is to be passed to the ARR upon error. The address is placed in the first four bytes of the area pointed to by SDWAPARM and in GPR 2. Note that the second four bytes of the area pointed to by SDWAPARM will not contain interface information.

To code: Specify the RX-type address, or address in register (2)-(12), of the parameter area.

,ARRPARAMPTR64=*arrparamptr64*
When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that contains the address of the parameter area that is to be passed to the ARR upon error. The address is placed in the 8-byte area pointed by SDWAPARM and in the 64-bit GPR 2.

To code: Specify the RX-type address, or address in register (2)-(12), of a 64-bit pointer field.

,ARRPARAM64=*arrparam64*
When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that is the parameter area that is to be passed to the ARR upon error. The address is placed in the 8-byte area pointed by SDWAPARM and in the 64-bit GPR 2.

To code: Specify the RX-type address, or address in register (2)-(12), of the parameter area.

,PARAMPTR=*paramptr*
When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is not in effect, a required input parameter that contains the address of a parameter that is to be passed to the target routine in GPR 1.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.

,PARAM=*param*
When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is not in effect. a required input parameter that is the parameter that is to be passed to the target routine in GPR 1.

To code: Specify the RX-type address, or address in register (2)-(12), of the parameter.

,PARAMPTR64=*paramptr64*
When DYNSTORAGE=AVAIL is in effect and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that contains the address of the parameter that is to be passed to the target routine in 64-bit GPR 1.

To code: Specify the RX-type address, or address in register (2)-(12), of a 64-bit pointer field.

,PARAM64=*param64*
When DYNSTORAGE=NOTAVAIL is specified and SYSSTATE AMODE64=YES is in effect, a required 8-byte input parameter that is to be passed to the target routine in 64-bit GPR 1.

To code: Specify the RX-type address, or address in register (2)-(12), of the parameter.

,TARGETPTR=*targetptr*
When DYNSTORAGE=AVAIL is in effect, a required input parameter that contains the address of the routine to which the system is to branch after establishing the ARR. The target routine will get control in the same key and state as the IEAARR caller, in AMODE 31, with the following input registers:

General Purpose Registers:

Register
Contents

0
Not part of the intended interface

1
Address of parameter area provided by IEAARR caller

2-13
Unchanged from the IEAARR caller

14
Tne return address

15
The address of the target routine

Access Registers:

Register
Contents

0-1
Not part of the intended interface

2-13
Unchanged from the IEAARR caller

14
Not part of the intended interface

15
Not part of the intended interface

The target routine gets control with one more entry on the linkage stack than existed when IEAARR was called. That linkage stack entry contains the caller's registers 2-13 which can be extracted using the EREG instruction if needed.

The target routine need not save any registers, but is expected to return to the address provided in GPR 14 on entry. The target routine can pass information back to the caller of IEAARR by placing it in GPR/AR 0, 1, and/or 15. The IEAARR caller will resume immediately after the IEAARR macro expansion.

The target routine gets control with its primary and secondary ASNs, which are both the same as the primary ASN when IEAARR was invoked.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.

,TARGET=*target*
When DYNSTORAGE=NOTAVAIL is specified. a required input parameter that is the routine to which the system is to branch after establishing the ARR. The target routine interface is identical to that described under the TARGETPTR parameter.

To code: Specify the RX-type address, or address in register (2)-(12), of the target routine.

,TARGETSTATE=PROB
,TARGETSTATE=SUP
A required parameter that indicates the requested PSW state of the target routine.

,TARGETSTATE=PROB
Indicates that the target routine is to get control in problem state. This should only be used by a caller currently in problem state.

,TARGETSTATE=SUP
Indicates that the target routine is to get control in supervisor state. This should only be used by a caller currently in supervisor state.

,ASYNCH=NO
,ASYNCH=YES
An optional parameter that identifies the ASYNCH attribute for the ARR. The definition of the ASYNCH attribute can be found in ESTAE and ESTAEX. The default is ASYNCH=NO.

,ASYNCH=NO
Indicates that the ARR is to have the ASYNCH=NO attribute.

,ASYNCH=YES
Indicates that the ARR is to have the ASYNCH=YES attribute.

ABEND codes

The caller may get the following abend code:

0C2-02
TARGETSTATE=SUP was requested by a caller currently running in problem state.

Return codes

None.

Example

Branch to the target routine pointed to by field TP, and establish as an ARR the routine pointed to by field AP. Pass to the target area in register 1 the contents of field PP. Make sure that the ARR will get access to the contents of field APP (which ordinarily would contain the address of a parameter area). Make sure that the target routine gets control in problem state (which implies that the caller of IEARR should currently be running in problem state).

The code is as follows.

```
IEAARR TARGETPTR=TP,ARRPTR=AP,PARAMPTR=PP,
      ARRPARAMPTR=APP,TARGETSTATE=PROB
   ...
```

Chapter 38. IEAFP — Floating point services

Description

IEAFP STOP allows you to request that, for your work unit, the system stop its status saving of the additional floating point status provided with the binary floating point architecture. This status consists of the additional floating point (AFP) registers (FPRs 1, 3, 5, 7-15) and the floating point control (FPC) register. It will also stop its saving of the vector register status.

You would typically use this service only when you are a server task which "subdispatches" unrelated units of work (for instance, CICS transactions). To avoid subsequent units of work being penalized by the floating point actions of previous units of work, the additional FP status saving function of the operating system can be turned off. When a unit of work actually begins to use FP, all appropriate status saving will be resumed.

IEAFP START allows you to request that, for your work unit, the system prepare for subsequent saving of vector register status. You would typically use this if your work unit is an SRB or (for a task) if your first usage of vector registers might be while disabled for I/O and external interrupts or while holding a lock other than the LOCAL lock.

IEAFP STOPVECTOR is similar to IEAFP STOP but affects only the vector register status saving. It leaves unchanged the floating point register status saving.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8 - 15.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31-bit.
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	The caller is not required to hold any locks on entry. For STOP or STOPVECTOR, the caller may hold the local, CMS, or CPU lock. For START, the caller may hold the LOCAL lock (not a CML).
Control parameters:	None.

Programming requirements

The caller can include the IHAFPRET mapping macro to get equate symbols for the return and reason codes provided by the IEAFP macro.

Restrictions

IEAFP must not be issued from an asynchronous exit routine.

Input register information

Before issuing the IEAFP macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, when GPR 15 is non-zero

1
Used as a work register by the system

2 - 13
Unchanged

14 - 15
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0 - 1
Used as work registers by the system

2 - 13
Unchanged

14 - 15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IEAFP macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEAFP.
IEAFP	

Syntax	Description
␣	One or more blanks must follow IEAFP.
START	
STOP	
STOPVECTOR	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15), (REG15), or (R15).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (0) or (2) - (12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEAFP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

START
STOP
STOPVECTOR
A required input parameter.

START
A parameter keyword that indicates to prepare for saving additional floating point status. and vector status Resources will be allocated at this point so that initial use may occur with fewer restrictions.

To code: Specify a value.

STOP
A parameter keyword that indicates to stop saving additional floating point status and vector status until such time as a new operation requires it. Resources will not be freed until termination of the work unit, making re-starting faster.

To code: Specify a value.

STOPVECTOR
A parameter keyword that indicates to stop saving vector register status until such time as a new vector operation requires it. Saving of additional floating point status will continue.

To code: Specify a value.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

ABEND codes

None.

Return and reason codes

When the IEAFP macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro IHAFPRET provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the *xxxx* value.

Table 45. Return and reason codes for the IEAFP macro

Return code	Reason code	Equate symbol, meaning, and action
0	–	**Equate symbol**: IeafpRc_OK **Meaning**: IEAFP request successful. **Action**: None required.
8	–	**Equate symbol**: IeafpRc_InvParm **Meaning**: IEAFP request specifies parameters that are not valid. **Action**: Refer to the action provided with the specific reason code.
8	*xxxx*0801	**Equate symbol**: IeafpRsnBadFunction **Meaning**: Incorrect value passed to target routine. **Action**: Check for possible storage overlay.
C	–	**Equate symbol**: IeafpRc_Env **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	*xxxx*0C01	**Equate symbol**: IeafpRsnFromAsynchExit **Meaning**: IEAFP was issued from an asynchronous exit routine. **Action**: Avoid issuing IEAFP from an asynchronous exit routine.
C	*xxxx*0C02	**Equate symbol**: IeafpRsnFromNonPreemptibleSRB **Meaning**: IEAFP START was issued from an SRB that was a non-preemptible SRB. **Action**: Avoid issuing IEAFP from a non-preemptible SRB.

Table 45. Return and reason codes for the IEAFP macro (continued)		
Return code	**Reason code**	**Equate symbol, meaning, and action**
C	*xxxx*0C03	**Equate symbol**: IeafpRsnFromNotBITCB **Meaning**: IEAFP START was issued from a task that was not the jobstep program task nor a subtask of that task. **Action**: When using IEAFP START from a task, do so only from the jobstep program task or a subtask of that task.
C	*xxxx*0C04	**Equate symbol**: IeafpRsnLocked **Meaning**: IEAFP START was issued when holding a lock other than the LOCAL lock. **Action**: Avoid using IEAFP START when holding a lock other than the LOCAL lock.
C	*xxxx*0C05	**Equate symbol**: IeafpRsnNoStorage **Meaning**: Necessary system storage could not be obtained. For a task mode request, it is LSQA storage. For an SRB mode request, it is ESQA storage. **Action**: If issued in task mode, use IEAFP START at an earlier time in the jobstep. If issued in SRB mode, arrange for some current user of ESQA storage to release it or run this after a re-IPL after ensuring additional ESQA storage availability for that IPL.
C	*xxxx*0C06	**Equate symbol**: IeafpRsnSuperBit **Meaning**: IEAFP START was issued from a work unit with at least one bit on in the PSASUPER word. A DIE is one such kind of work unit. **Action**: Avoid issuing IEAFP from a DIE or any other work unit running with PSASUPER on.

Examples

Example 1: Stop additional status saving.

The code is as follows:

```
         IEAFP STOP
```

Chapter 39. IEAGSF — Guarded-Storage Facility services

Environment

The requirements for the caller are:

Environmental Factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	For START and STOP: Any PASN, any HASN, any SASN. For UPDATE: PASN=HASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller is not required to hold any locks on entry, but may hold the LOCAL lock.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

Control parameters must be below the 2G line.

Include the IHAGSRET mapping macro to get equate symbols for the return and reason codes provided by the IEAGSF macro.

Restrictions

IEAGSF is only available if the CVTGSF bit is on.

IEAGSF must not be issued from an SRB or an asynchronous exit routine (IRB).

IEAGSF must not be issued for a task above the jobstep program TCB.

Supervisor-state or system key users must not use this facility if unauthorized programs can run in the same address space because unauthorized programs have the ability to change the GSF controls.

The caller must not have an FRR established when invoking IEAGSF UPDATE.

Input Register Information

Before issuing the IEAGSF macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output Register Information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, when GPR 15 is non-zero

1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

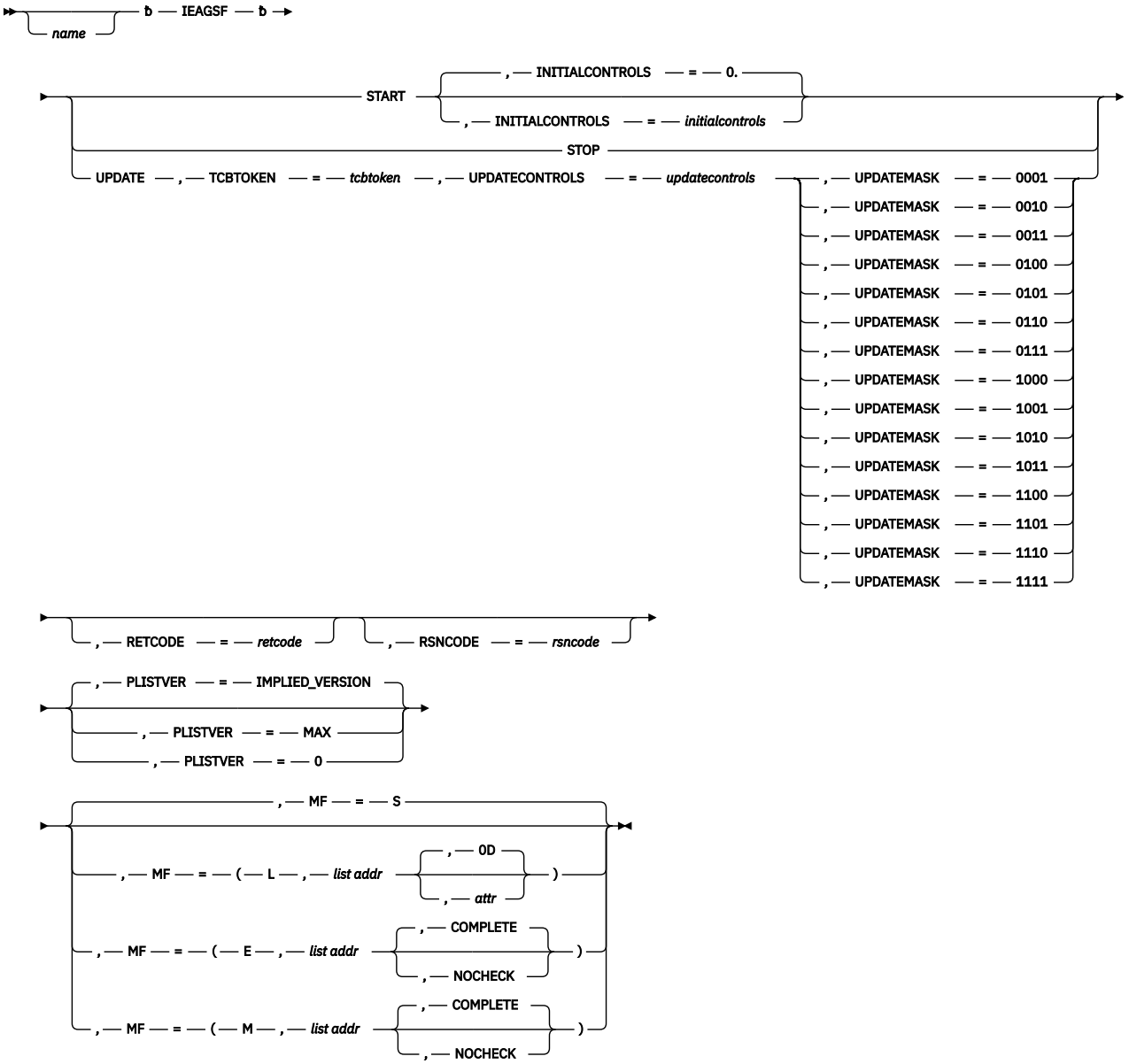
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

None.

Syntax

main diagram



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEAGSF macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,INITIALCONTROLS=*initialcontrols*
,INITIALCONTROLS=0.
When START is specified, an optional input parameter, which contains the initial GSF controls (GSCB) to be used. See the description of the Guarded-Storage Control Block in the z/Architecture Principles of Operation manual. The default is 0. The initial GSCB will be set to zeros.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of IEAGSF in the following order:

- Use IEAGSF ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use IEAGSF ...MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
- Use IEAGSF ...MF=(E,list-addr,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters. When using the NOCHECK option, be sure that it is preceded by a modify or execute form invocation that specifies or defaults to the COMPLETE option. This ensures that the parameter list is initialized completely.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15). Do not specify with MF=M.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0). Do not specify with MF=M.

START
A required input parameter keyword that indicates that the system should start the Guarded-Storage Facility and start the saving of Guarded-Storage Facility control information for the current TCB.

To code: Specify a value.

,STOP
A required input parameter keyword that indicates that the system should stop the saving of Guarded-Storage Facility control information and stop the Guarded-Storage Facility for the current TCB. Note that MF= is not supported (or needed) for IEAGSF STOP.

To code: Specify a value.

,TCBTOKEN=*tcbtoken*
When UPDATE is specified, a required input parameter, which contains the TCBTOKEN of the task to be updated. All of that task's subtasks will also be updated. The TCBTOKEN must be valid and must represent the jobstep program task or a subtask of the jobstep program task.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,UPDATE
A required input parameter keyword that indicates that the system should synchronously update the Guarded-Storage Facility control information for the input TCB and all of its subtasks. Any tasks which have not issued IEAGSF START will be ignored.

To code: Specify a value.

,UPDATECONTROLS=*updatecontrols*
When UPDATE is specified, a required input parameter, which contains a new GSF Control Block ('GSCB') to be used. See the description of the Guarded-Storage Control Block in the z/Architecture Principles of Operation manual.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,UPDATEMASK=0001
,UPDATEMASK=0010
,UPDATEMASK=0011
,UPDATEMASK=0100
,UPDATEMASK=0101
,UPDATEMASK=0110
,UPDATEMASK=0111
,UPDATEMASK=1000
,UPDATEMASK=1001
,UPDATEMASK=1010
,UPDATEMASK=1011
,UPDATEMASK=1100
,UPDATEMASK=1101
,UPDATEMASK=1110
,UPDATEMASK=1111

When UPDATE is specified, a required parameter, which specifies a mask that describes which of the 4 doublewords of each task's GSCB are to be updated using the new GSCB specified by UPDATECONTROLS. Use caution when updating the fourth doubleword - you may not want multiple tasks sharing the GSEPL that it points to.

,UPDATEMASK=0001
specifies that only the fourth doubleword is to be updated.

,UPDATEMASK=0010
specifies that only the third doubleword is to be updated.

,UPDATEMASK=0011
specifies that only the third and fourth doubleword are to be updated.

,UPDATEMASK=0100
specifies that only the second doubleword is to be updated.

,UPDATEMASK=0101
specifies that only the second and fourth doubleword are to be updated.

,UPDATEMASK=0110
specifies that only the second and third doublewords are to be updated.

,UPDATEMASK=0111
specifies that only the second, third, and fourth doublewords are to be updated.

,UPDATEMASK=1000
specifies that only the first doubleword is to be updated.

,UPDATEMASK=1001
specifies that only the first and fourth doublewords are to be updated.

,UPDATEMASK=1010
specifies that only the first and third doublewords are to be updated.

,UPDATEMASK=1011
specifies that only the first and third and fourth doublewords are to be updated.

,UPDATEMASK=1100
specifies that only the first and second doublewords are to be updated.

,UPDATEMASK=1101
specifies that only the first and second and fourth doublewords are to be updated.

,UPDATEMASK=1110
specifies that only the first and second and third doublewords are to be updated.

,UPDATEMASK=1111
specifies that all four doublewords are to be updated.

ABEND Codes

None.

Return and Reason codes

When the IEAGSF macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro IHAGSRET provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the *xxxx* value.

Table 46. Return and reason codes for the IEAGSF macro

Return code	Reason code	Equate symbol, meaning and action
0	–	**Equate symbol**: IeaGsfRc_OK **Meaning**: IEAGSF request successful. **Action**: None required.
8	–	**Equate symbol**: IeaGsfRc_InvParm **Meaning**: IEAGSF request specifies parameters that are not valid. **Action**: Refer to the action provided with the specific reason code.
8	*xxxx*0801	**Equate symbol**: IeaGsfRsnBadFunction **Meaning**: Incorrect value passed to target routine. **Action**: Check for possible storage overlay.
8	*xxxx*0802	**Equate symbol**: IeaGsfRsnUpdateTcbBad **Meaning**: The TCB specified by the TTOKEN for an IEAGSF UPDATE request is no longer valid, or is terminating, or is neither the jobstep program task nor a subtask of the jobstep program task. **Action**: N/A
C	–	**Equate symbol**: IeaGsfRc_Env **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	*xxxx*0C01	**Equate symbol**: IeaGsfRsnFromAsynchExit **Meaning**: IEAGSF was issued from an asynchronous exit routine. **Action**: Avoid issuing IEAGSF from an asynchronous exit routine.
C	*xxxx*0C02	**Equate symbol**: IeaGsfRsnFromSRB **Meaning**: IEAGSF was issued from an SRB. **Action**: Avoid issuing IEAGSF from an SRB.

Table 46. Return and reason codes for the IEAGSF macro (continued)		
Return code	**Reason code**	**Equate symbol, meaning and action**
C	*xxxx*0C03	**Equate symbol**: IeaGsfRsnFromNotBITCB **Meaning**: IEAGSF was issued from a task that was neither the jobstep program task nor a subtask of the jobstep task. **Action**: Only issue IEAGSF from the jobstep program task or a subtask of that task.
C	*xxxx*0C04	**Equate symbol**: IeaGsfRsnLocked **Meaning**: IEAGSF was issued while holding a lock other than the LOCAL lock. **Action**: Avoid issuing IEAGSF while holding a lock other than the LOCAL lock.
C	*xxxx*0C05	**Equate symbol**: IeaGsfRsnNoStorage **Meaning**: Necessary system storage could not be obtained from LSQA. **Action**: Use IEAGSF START at an earlier time in the jobstep.
C	*xxxx*0C06	**Equate symbol**: IeaGsfRsnSuperBit **Meaning**: IEAGSF was issued from a work unit with at least one bit on in the PSASUPER word. **Action**: Avoid issuing IEAGSF from a work unit running with a PSASUPER bit on.
C	*xxxx*0C07	**Equate symbol**: IeaGsfRsnNotAvailable **Meaning**: The Guarded-Storage Facility is not available on this system. **Action**: Ensure that the hardware supports this facility.
C	*xxxx*0C08	**Equate symbol**: IeaGsfRsnUpdateInXM **Meaning**: IEAGSF UPDATE has been called with a Primary address space that is different than the Home address space. **Action**: Only invoke IEAGSF UPDATE with the Primary address space equal to the Home address space.

Example

Start using the Guarded-Storage Facility with an initial GSCB:

The code is as follows.

```
IEAGSF START,INITIALCONTROLS=MYGSCB
```

Chapter 40. IEALSQRY — Linkage stack query

Description

The linkage stack query macro IEALSQRY checks the level of the current entry on the linkage stack relative to the level of the entry associated with the most recent recovery routine. The output of the macro is a value (in the TOKEN parameter) a recovery routine can use to ensure that a retry routine runs with the appropriate linkage stack entry. If the return code is not zero, the value in TOKEN is not valid.

Your program is to pass the value in TOKEN to a recovery routine. When the recovery routine gets control, it can place that value in the SDWA field SDWALSLV. That action ensures that, when a retry routine gets control, it has the correct linkage stack level. For information about how to use the value in TOKEN, see the topic about the linkage stack at a retry routine in *z/OS MVS Programming: Authorized Assembler Services Guide*.

The output of IEALSQRY depends upon the current environment and on the recovery environment that exists:

- If FRRs exist, the value returned in TOKEN is the difference between the current level of the linkage stack and the level of the stack at the time the FRR was activated.
- If no FRRs exist, but the caller holds a lock or is in SRB mode, a return code of 8 is returned.
- If no FRRs exist, and the caller is unlocked and in task mode, and at least one ESTAE-type recovery routine is in effect, the output depends on the most recently activated routine:
  - If it is a STAE, STAI, or FESTAE routine, a return code of 8 is returned.
  - If it is an ARR, the value returned in TOKEN is the difference between the current level of the linkage stack and the level of the stack at the time the ARR was activated.
  - If it is an ESTAE or ESTAEX for the current RB, the value returned is the difference between the current level of the linkage stack and the level of the stack at the time the ESTAE or ESTAEX was activated.
  - If it is an ESTAI, the value returned is the difference between the current level of the linkage stack and the level of the stack at the time the newest PRB that is older than the oldest non-PRB was created (or simply the newest PRB if all the RBs are PRBs).
- If no FRRs exist, and the caller is unlocked and in task mode, and no ESTAEXs, ESTAEs, STAEs, or FESTAEs exist for this RB and no ESTAIs, STAIs, or ARRs in effect, a return code of 8 is returned.

See *z/OS MVS Programming: Authorized Assembler Services Guide* for further information about the use of the SDWALSLV field.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, PSW key 8-15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
Amode:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts

Environmental factor	Requirement
Locks:	The caller can hold the local lock of the primary address space and can additionally hold the CMS lock. The caller can hold the CPU lock. No locks are required. If the primary address space does not match the home address space, the caller must not hold the local lock of the home address space.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

Do not issue the IEALSQRY macro in a DIE routine.

Input register information

Before issuing the IEALSQRY macro, the caller does not have to place any information into a general purpose register (GPR) or access register (AR).

Output register information

When control returns to the caller from IEALSQRY, the GPRs contain:

Register
Contents

0
Output token value, which is copied to the area specified by the TOKEN parameter.

1
Used as a work register by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Return code.

When control returns to the caller from IEALSQRY, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14 and 15
Used as work registers by the system.

Performance implications

This macro should not be used in a performance-sensitive program.

Syntax

The standard form of the IEALSQRY macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEALSQRY.
IEALSQRY	
ƀ	One or more blanks must follow IEALSQRY.
	Valid parameters
TOKEN=*token*	*token*: RS-type address or register (1) - (12). **Default**: Leave token in GPR 0.
,RETCODE=*retcode*	*retcode*: RS-type address, or register (2) - (12). **Default**: No retcode processing.

The parameters are explained as follows:

TOKEN=*token*
Specifies a halfword area (or the address of the area in register (1)-(12)) where the system places a value that indicates the difference between the number of linkage stack entries present when the recovery routine was activated and the number that are currently present. A recovery routine can place this value in field SDWALSLV (in mapping macro IHASDWA) to ensure that the retry routine runs with the proper level of the linkage stack. If you do not use TOKEN, you can find the value in GPR 0.

RETCODE=*retcode*
Specifies a fullword output variable (or register (2)-(12)) into which the system copies the return code GPR 15. If you do not use RETCODE, you can find the return code in GPR 15.

ABEND codes

The IEALSQRY caller might receive abend code X'B78'. For detailed abend code information, see *z/OS MVS System Codes*.

Return codes

When control returns to the caller, register 15 contains one of the following decimal return codes (hexadecimal values are shown in parentheses):

Table 47. Return Codes for IEALSQRY

Return Code	Meaning and Action
0 (0)	**Meaning**: Successful completion. A valid value is in the TOKEN parameter. **Action**: None required.

Table 47. Return Codes for IEALSQRY (continued)	
Return Code	**Meaning and Action**
4 (4)	**Meaning**: The system encountered a linkage stack entry that violates the authorization or stacking-PC conditions that are required for successful retry. **Action**: Avoid using the token when retrying. You cannot retry to the current linkage stack level.
8 (8)	**Meaning**: No recovery routine of the proper type exists. If in a state from which you cannot issue ESTAEX, no FRR exists. If in a state from which you can issue ESTAEX, either no recovery routine exists or the most recently activated recovery routine is STAE, STAI, or FESTAE. **Action**: Avoid using the token when retrying. You cannot retry to the current linkage stack level.
12 (C)	**Meaning**: You called IEALSQRY in a DIE routine. **Action**: Do not use the IEALSQRY macro in a DIE routine.
16 (10)	**Meaning**: System error. **Action**: Report the problem to IBM. Avoid using the token when retrying. You cannot retry to the current linkage stack level.

Example

Obtain the value that a recovery routine can place in SDWALSLV:

```
         IEALSQRY TOKEN=MYTOKEN
  .
  .
MYTOKEN  DS    H        Output TOKEN
```

Chapter 41. IEAMETR — Query external time reference status

Description

IEAMETR can be used to query external time reference (ETR) status and connection information for the current MVS image. This information is returned in the output area specified by the OUTAREA keyword.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Any state or key
Dispatchable unit mode:	Either Task, SRB, or DIE mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	Any lock may be held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IEAMETR macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IEAMETR macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The IEAMETR macro is written as follows:

Syntax	Description
xlabel	*xlabel*: Optional symbol. Begin *xlabel* in column 1. The name must conform to the rules for an ordinary assembler language symbol. DEFAULT: No name.
ƀ	One or more blanks must precede IEAMETR.
IEAMETR	
ƀ	One or more blanks must follow IEAMETR.
OUTADDR=*xoutaddr*	
,MF=S	**Default**: S
,MF=(L,xmfctrl,xmfattr | 0D)	
,MF=(E,xmfctrl,COMPLETE)	

Parameters

The parameters are explained as follows:

OUTADDR=*xoutaddr*
A required input parameter that contains the address of the 24-byte output area to receive the output. The area is mapped by macro IHAETRI.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,MF=S | L | E
Optional keyword input that specifies the macro form.

S
Specifies the standard form of the macro. Generates code to put the parameters into an in-line parameter list and invoke the desired service. Full checking for required macro keys is done along with supplying defaults for omitted optional parameters.

DEFAULT: S

L
Specifies the list form of the macro. Defines an area to be used for the parameter list. Any other macro parameters are flagged as errors.

E
Specifies the execute form of the macro. Generates code to put the parameters into the parameter list specified by xmfctrl and provides syntax checking with default setting.

,xmfctrl
Required input. It is the name of a storage for the parameter list.

,xmfattr | 0D
Optional 60 character input string that varies from 1 to 60 characters. It can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list.

DEFAULT: 0D. 0D forces the parameter list to a doubleword boundary.

,xmfctrl
Required input. It is the name (RS-type), or address in register (1)-(12), of a storage area for the parameter list.

,COMPLETE
Optional keyword input that specifies the degree of macro parameter syntax checking.

DEFAULT: COMPLETE. Checking for required macro keys is done, and defaults are supplied for omitted optional parameters.

Return codes

Table 48. Return Codes for the IEAMETR Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: ETR status and port data was successfully obtained. **Action**: None.
04	**Meaning**: ETR status information is available, but port is not. **Action**: None.
08	**Meaning**: No status or port data is available. **Action**: None.
0C	**Meaning**: The parameter list is not in the user's primary address space. **Action**: Use a parameter list in the primary address space.

Chapter 42. IEAMRMF3 — Obtain address space dispatchability data

Description

The IEAMRMF3 macro provides information about the dispatchability of address spaces. Use IEAMRMF3 to determine which address spaces are currently running on a processor and which address spaces are waiting for a processor. To get information about the dispatchability of enclaves, use the IWMRQRY macro. IMWRQRY is described in *z/OS MVS Programming: Workload Management Services*.

The output you receive from this macro contains an array of elements, with each element representing an address space. For each address space, the system indicates that the address space is one of the following:

- Dispatchable and running on a processor
- Dispatchable and waiting to run on a processor
- Not dispatchable.

The number of elements you receive is the maximum number of address spaces in the system.

Use the IEAMRMF3 macro for monitoring your system. Typically, a monitoring program issues the macro repeatedly to obtain samples over a period of time. For some invocations of the macro, the system might be unable to retrieve the data, and the caller receives a return code of 4. However, this is generally a temporary condition; if the caller was issuing the macro repeatedly, the caller should continue to do so and should receive data on subsequent invocations. If the caller receives return code 4 several times in succession, the caller should stop issuing the macro. How many times the caller issues the macro after a return code of 4 is up to the installation.

Under certain conditions, the system abnormally ends the caller of IEAMRMF3 with an X'0C4' abend code. The caller must supply its own recovery routine to capture this abend code and retry.

Environment

Requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held, except the CPU lock if the caller is disabled for I/O and external interrupts.
Control parameters:	Must be in the primary address space

Programming requirements

The caller must obtain storage for the output returned by this macro. See the OUTAREA parameter for further information.

Include the following mapping macros in the module that calculates the size of the storage area for the output:

- IHADSD, which maps the DSD data area
- IHAASVT, which maps the ASVT data area
- CVT, which maps the CVT data area
- IHAPSA, which maps the PSA data area

If a separate module examines the data returned by IEAMRMF3, that module must also include the IHADSD mapping macro.

For the mappings provided by the IHAASVT, CVT, IHADSD, and IHAPSA mapping macros, see the information in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Restrictions

None.

Register information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the caller issued the macro. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

On entry to IEAMRMF3, general purpose register (GPR) 13 must contain the address of a 72-byte standard save area. If the caller is disabled, the save area must be pagefixed.

When control is returned to the calling program, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control is returned to the calling program, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The standard form of the IEAMRMF3 macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEAMRMF3.
IEAMRMF3	
ƀ	One or more blanks must follow IEAMRMF3.
OUTAREA=*area name*	*area name*: Symbol.

Parameters

The parameter is explained as follows:

OUTAREA=*area name*
The required parameter that specifies the name of the area of storage to contain the output from the macro. The output contains an array of elements; each element represents an address space. The output is mapped by the IHADSD mapping macro.

The caller must obtain storage for *area name* on a fullword boundary, in the caller's primary address space. The caller is not required to initialize *area name*. If the caller is disabled, *area name* must be pagefixed.

Before issuing IEAMRMF3, you need to determine the size of the output area and obtain storage for it. To do so, use the following formula to determine the length of an element:

```
(DSDRSVD - DSDELEM) + (length of DSDRSVD)
```

Then, use that value in the following formula to determine the total size of *area name*:

```
(ASVTMAXU * length of an element) + (length of DSDFIXED)
```

ASVTMAXU is a field in the ASVT data area. DSDFIXED, DSDRSVD, and DSDELEM are fields in the DSD data area.

For each element representing an address space, the following are true:

- If the DSDUSING bit in the DSD is on, the address space is dispatchable and running on a processor.
- If the DSDWAIT bit in the DSD is on, the address space is dispatchable and is waiting to run on a processor.
- If neither the DSDUSING bit nor the DSDWAIT bit in the DSD is on, then either the address space is not dispatchable or the element does not represent a valid address space.

Return codes

When control returns from IEAMRMF3, GPR 15 contains one of the following return codes:

Table 49. Return Codes for the IEAMRMF3 Macro	
Hexadecimal Return Code	**Meaning and Action**
00	**Meaning**: Data successfully collected. **Action**: None.
04	**Meaning**: The system was not able to gather the data on this invocation of the macro. **Action**: Reissue the macro. If you receive return code 4 several times in succession, you need to stop issuing the macro and inform your technical support personnel.

Example

Issue the IEAMRMF3 macro to obtain address space dispatchability data. The caller in this example is enabled for I/O and external interrupts, and is APF-authorized.

In this example, the caller issues IEAMRMF3 only once. If the return code from IEAMRMF3 is zero, the caller loops through the elements to look at the data, and does not issue the macro again. If the return code is not zero, the caller does not make another attempt to obtain data. A more typical scenario would be to issue the macro repeatedly if the return code is zero to obtain data over a period of time, and to issue the macro again even if the return code is 4, hoping to obtain data on a subsequent invocation. This example is intended only as an illustration of how to issue the macro, and an illustration of one way you might look at the data returned by the macro.

```
MONITOR  CSECT
MONITOR  AMODE 31
MONITOR  RMODE ANY
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
*
         USING *,R15               Establish addressability
*
         STM     R14,R12,12(R13)   Save registers in caller's save
*                                  area
         DROP    R15
         LR      R12,R15           Copy base register R12 because R15
*                                  is volatile across interfaces used
*                                  in this module
@PSTART  EQU     MONITOR
         USING   @PSTART,R12       Reestablish addressability using
*                                  R12 as base register
*****************************************************************
*
*   Invoke the IEAMRMF3 service to obtain address space
*   dispatchability data.
*
*****************************************************************
*
*   Change to key 0, supervisor state
*
         MODESET KEY=ZERO,MODE=SUP
*
*   Obtain storage for a register save area to pass to IEAMRMF3
*
         LA      R0,72
         GETMAIN RU,LV=(R0)
         LR      R4,R13              Save the caller's save area
*                                    address
         LR      R13,R1              Get address of save area to
```

```
*                                    pass to IEAMRMF3
*
*  Calculate the size of the storage to obtain for the output from
*  IEAMRMF3:
*  (ASVTMAXU * length of an element) + (length of DSDFIXED)
*
```

```
         USING   PSA,0
         L       R2,FLCCVT              Get pointer to the CVT from the
*                                       PSA
         L       R9,CVTASVT-CVTMAP(,R2) Get pointer to the ASVT from
*                                       the CVT
         L       R9,ASVTMAXU-ASVT(,R9)  Get ASVTMAXU
         LA      R11,ELEMSIZE           Get the length of an element
         MR      R8,R11                 Multiply
         LA      R6,L'DSDFIXED          Get the length of DSDFIXED
         AR      R9,R6                  Add to get total
*
*   Obtain storage for the DSD data area
*
         GETMAIN RU,LV=(R9)
*
* Issue the IEAMRMF3 macro to return address space dispatchability data
* Note: Register 13 contains the address of the 72-byte save area to
* pass to IEAMRMF3.
*
         LR      R6,R1                  Move address of storage that
*                                       was just obtained into R6.
*
         USING   DSD,R6                 Map the DSD on the storage area
*                                       that was just obtained.
*
         IEAMRMF3 OUTAREA=DSD           Issue the IEAMRMF3 macro passing
*                                       the DSD data area to be used for
*                                       the output.
*
         LTR     R15,R15                Check return code from IEAMRMF3.
         BNZ     NODATA                 For a nonzero return code,
*                                       do not attempt to look at data.
*
* Look at the elements that are filled in by using the DSDINDXF field
* to find the first element, and the DSDINDXN field to chain to
* the next element.
*
         LH      R11,DSDINDXF           Get the index of the first entry
*                                       that is filled in. If the value
*                                       is x'FFFF' then no entries are
*                                       filled in.
         L       R7,DSDAPTR             Get the address of the array
NEXTELEM DS      0H
         CH      R11,=X'FFFF'           If the index is x'FFFF' then
*                                       this is the last element that is
*                                       filled in.
         BE      ALLDONE                There are no more elements to
*                                       process.
         BCTR    R11,0                  Decrement the index by 1.  The
*                                       entry for ASID 1 is the first
*                                       entry.
         LA      R2,ELEMSIZE            Get the element size
         MR      R10,R2                 Multiply the index by the
*                                       element size.
         AR      R11,R7                 Add the array pointer and the
*                                       result to obtain the address of
*                                       the entry that we want to look at
         USING   DSDELEM,R11
*        .                              This area contains whatever code
*        .                              the routine uses to look at the
*        .                              required fields.
*
```

```
         LH      R11,DSDINDXN           Obtain the index of the next
*                                       entry to look at.
         DROP    R11
         B       NEXTELEM               Go process the next element
*
NODATA   DS      0H
         LR      R2,R15                 Save the nonzero return code
*                                       from IEAMRMF3 in R2.  R15 is
*                                       volatile across the interfaces.
         B       FREESTOR
```

```
ALLDONE  DS      0H
*
*   Set a return code of zero.
*
         LA      R2,0                Save the zero return code that
*                                    this module sets in R2.  R15 is
*                                    volatile across the interfaces.
*
FREESTOR DS      0H
*
*   Free the storage for the register save area passed to IEAMRMF3.
*
         LA      R0,72
         FREEMAIN RU,LV=(R0),A=(13)
*
*   Restore the caller's save area address
*
         LR      R13,R4
*
*   Free the storage for the DSD data area
*
         FREEMAIN RU,LV=(R9),A=(6)
         DROP    R6                  Drop addressability to the
*                                    DSD data area.
*
*   Change to problem state, not key 0
*
         MODESET KEY=NZERO,MODE=PROB
*
         LR      R15,R2              Copy the return code to
*                                    R15.
         L       R14,12(R13)
         LM      R0,R12,20(R13)
         BR      R14     Return to the caller
         DROP    R12
*
*  Equate for length of an element:
*
ELEMSIZE EQU     DSDRSVD-DSDELEM+L'DSDRSVD
         EJECT
         IHAASVT LIST=NO             Mapping macro for the ASVT
         EJECT
         CVT     DSECT=YES,LIST=NO    Mapping macro for the CVT
         EJECT
         IHADSD  LIST=YES            Mapping macro for the DSD
         EJECT
         IHAPSA  LIST=NO             Mapping macro for the PSA
         END     MONITOR
```

Chapter 43. IEAMSCHD — Schedule an SRB

Description

Use the IEAMSCHD macro to schedule a service request block (SRB) for asynchronous execution. When you schedule an SRB, you can specify dispatching priority and processor affinity. Preemptable SRBs (PRIORITY=CLIENT,PRIORITY=ENCLAVE, or PRIORITY=PREEMPT) can also be scheduled with a minor priority.

Optionally, the scheduling program can specify:

- A functional recovery routine (FRR)
- A resource manager termination routine (RMTR)

  The scheduling program can specify an RMTR to be invoked by the PURGEDQ service. The RMTR is responsible for cleaning up resources on behalf of an SRB routine if it has been purged by PURGEDQ before it is dispatched.

IBM recommends using IEAMSCHD rather than the SCHEDULE macro. For information about how to schedule an SRB, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	PSW key 0-7, or supervisor state with any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any HASN, any PASN, any SASN
AMODE:	31-bit
ASC mode:	Primary, or access register
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any. A caller who specifies SYNCH=YES cannot hold any locks.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

- If the caller specifies RMTRADDR, the resource manager termination routine must reside in MVS common storage.
- If the caller specifies PRIORITY=ENCLAVE, the enclave token must have been previously obtained through the IWMECREA macro.

Restrictions

- Address space resource managers cannot use the STOKEN parameter.
- If you issue IEAMSCHD from a set DIE routine, you cannot specify PRIORITY=CURRENT, you cannot specify SYNCH=YES, and the DIE routine must be running in supervisor state, with PSW key 0.
- If your program specifies SYNCH=YES and the scheduled SRB issues the SRBSTAT SAVE macro or invokes any services that issue SRBSTAT SAVE, control returns to your program immediately.

Input register information

Before issuing the IEAMSCHD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IEAMSCHD macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEAMSCHD macro.
IEAMSCHD	
␣	One or more blanks must follow IEAMSCHD macro.

Syntax	Description
EPADDR=*epaddr*	*epaddr*: RS-type address or register (2) - (12).
,ENV=HOME	**Default**: HOME
,ENV=PRIMARY	
,ENV=FULLXM	
,ENV=STOKEN	
,TARGETSTOKEN=*targetstoken*	*targetstoken*: RS-type address or register (2) - (12).
,FEATURE=NONE	**Default**: NONE
,FEATURE=CRYPTO	
,PRIORITY=LOCAL	**Default**: LOCAL
,PRIORITY=GLOBAL	
,PRIORITY=CURRENT	
,PRIORITY=PREEMPT	
,PRIORITY=CLIENT	
,PRIORITY=ENCLAVE	
,MINORPRIORITY=ZERO	**Default**: ZERO
,MINORPRIORITY=*minorpriority*	*minorpriority*: RS-type address or register (2) - (12).
,SRBIDTOKEN=*token*	*token*: 16-byte output area.
,CLIENTSTOKEN=*clientstoken*	*clientstoken*: RS-type address or register (2) - (12).
,ENCLAVETOKEN=*enclavetoken*	*enclavetoken*: RS-type address or register (2) - (12).
,PARM=ZERO	**Default**: ZERO
,PARM=*parm*	*parm*: RS-type address or register (2) - (12).
,FRRADDR=NOFRR	**Default**: NOFRR
,FRRADDR=*frraddr*	*frraddr*: RS-type address or register (2) - (12).

Syntax	Description
,SDWALOC31=NO	**Default**: NO
,SDWALOC31=YES	
,KEYVALUE=INVOKERKEY	**Default**: INVOKERKEY
,KEYVALUE=*keyvalue*	*keyvalue*: RS-type address or register (2) - (12).
,LLOCK=NO	**Default**: NO
,LLOCK=YES	
,RMTRADDR=NORMTR	**Default**: NORMTR
,RMTRADDR=*rmtraddr*	*rmtraddr*: RS-type address or register (2) - (12).
,PURGESTOKEN=NOPSTOKEN	**Default**: NOPSTOKEN
,PURGESTOKEN=*purgestoken*	*purgestoken*: RS-type address or register (2) - (12).
,PTCBADDR=NOPTCB	**Default**: NOPTCB
,PTCBADDR=*ptcbaddr*	*ptcbaddr*: RS-type address or register (2) - (12).
,FLAGS=NO_FLAGS	**Default**: NO_FLAGS
,FLAGS=*flags*	*flags*: RS-type address or register (2) - (12).
,SYNCH=NO	**Default**: NO
,SYNCH=YES	
,SYNCHCOMPADDR=NOVALUE	**Default**: NOVALUE
,SYNCHCOMPADDR=*compaddr*	*compaddr*: RS-type address or register (2) - (12).
,SYNCHCODEADDR=*codeaddr*	*codeaddr*: RS-type address or register (2) - (12).
,SYNCHRSNADDR=*rsnaddr*	*rsnaddr*: RS-type address or register (2) - (12).
,TRANSFER=NO	**Default**: NO
,TRANSFER=YES	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).

Syntax	Description
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,PLISTVER=1	
,PLISTVER=2	
,PLISTVER=3	
,PLISTVER=4	
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

EPADDR=*epaddr*
Specifies the address of the SRB routine to be scheduled for asynchronous execution.

Note: The SRB routine receives control in 31–bit addressing mode.

To code: Specify the name (RS-type), or address in register (2)-(12), of a required 4-byte input parameter.

,ENV=HOME
,ENV=PRIMARY
,ENV=FULLXM
,ENV=STOKEN
Optional input parameter that specifies the addressing and cross memory environment in which the SRB routine is to receive control.

HOME
Specifies that the SRB routine is to receive control in the current home address space.

Default: HOME

PRIMARY
Specifies that the SRB routine is to receive control in the current primary address space.

FULLXM
Specifies that the SRB routine is to receive:

- Control in the scheduling program's current cross memory environment
- A copy of the scheduling program's dispatchable unit-access list (DU-AL). For details about how the system copies a DU-AL, see the topic on access lists in *z/OS MVS Extended Addressability Guide*.

This provides the SRB routine with addressability to the same address spaces and data spaces as the scheduling program.

STOKEN
Specifies that the SRB routine is to receive control in the address space specified by TARGETSTOKEN=*targetstoken*. If the target stoken passed is no longer valid, then the caller receives AC7 abend code with reason code X'00080001'.

,TARGETSTOKEN=*targetstoken*
Specifies the space token (STOKEN) of the address space in which the SRB routine is to receive control.

To code: Specify the name (RS-type), or address in register (2)-(12), of a required 64-bit input parameter.

,PRIORITY=LOCAL
,PRIORITY=GLOBAL
,PRIORITY=CURRENT
,PRIORITY=PREEMPT
,PRIORITY=CLIENT
,PRIORITY=ENCLAVE
Optional input parameter that specifies the priority at which the SRB routine is dispatched, and whether the SRB is to be preempted.

LOCAL
Schedules an SRB at a priority equal to that of the address space into which it was scheduled. With a LOCAL priority, an SRB has a higher priority than any task or preemptable SRB in that address space.

Default: LOCAL

GLOBAL
Schedules an SRB at a priority equal to the highest priority work in the system, regardless of the address space into which it was scheduled. An SRB scheduled with PRIORITY=GLOBAL is not preemptable.

CURRENT
Schedule an SRB at a priority equal to that of the scheduling work unit.

Task Mode Callers: For task mode callers, the SRB is always preemptable. If the task has joined an enclave, the SRB routine inherits the enclave's major priority and the task's minor priority. Otherwise, the SRB routine inherits the major priority of the task's home address space and the minor priority of the task. If the scheduling task and the scheduled SRB have different home address spaces, then the scheduled SRB is also converted to a client SRB.

Nonpreemptable SRB Mode Callers: For SRB mode callers that are not preemptable, the scheduled SRB inherits the PRIORITY option used to schedule the scheduling SRB routine:

- If PRIORITY=GLOBAL was used, the scheduled SRB will have a priority as high as the highest priority in the system.
- If PRIORITY=LOCAL was used, the scheduled SRB will have a priority that is higher than any task or preemptable SRB in the scheduled SRB's home address space.

Preemptable SRB Mode Callers: For preemptable SRB mode callers, the scheduled SRB is always preemptable. If the scheduling SRB was scheduled into an enclave, the scheduled SRB inherits the enclave's major priority and the scheduling SRB's minor priority. Otherwise, the scheduled SRB inherits the major priority of the scheduling SRB's home address space and the minor priority of the scheduling SRB. If the scheduling SRB and the scheduled SRB have different home address spaces, then the scheduled SRB is also converted to a client SRB.

PREEMPT
Schedules a preemptable SRB routine that inherits the major priority of the target home address space (the home address space as specified on the ENV parameter).

CLIENT
Schedules a preemptable SRB that inherits the major priority of the address space named by the STOKEN specified on the CLIENTSTOKEN parameter. The processor time used by this SRB is accumulated in the address space specified by the *clientstoken*.

ENCLAVE
Schedules a preemptable RB into an enclave. The SRB inherits the major priority of the enclave specified on the ENCLAVETOKEN keyword. The processor time used by this SRB is accumulated in the enclave specified by the *enclavetoken*.

,MINORPRIORITY=ZERO
,MINORPRIORITY=*minorpriority*
Specifies the minor priority to assign to the SRB routine. SRB routines with higher minor priority are dispatched before preemptable-class SRB routines and before tasks with lower minor priority in the same address space. A minor priority of X'00' is the lowest and X'FF' is the highest.

The minor priority parameter assigns the SRB routine a priority that is comparable to a task's dispatching priority in the address space. The caller can specify priorities for SRB routines so that they are dispatched before, with, or after tasks in the address space.

Default: ZERO

To code: Specify the name (RS-type), or address in register (2)-(12), of an 8-bit input parameter. MINORPRIORITY is optional for PRIORITY=PREEMPT, PRIORITY=CLIENT, and PRIORITY=ENCLAVE.

,SRBIDTOKEN=*token*
Specifies the name of an optional 16-byte output area where a token is placed to be used to fully identify the SRB to the system. The token is used to request termination of a preemptable SRB via CALLRTM TYPE=SRBTERM. The SRBIDTOKEN keyword may be used even if the program runs on a release for which the support is not provided or on a release on which the support is not installed. If the program is running on a release that supports SRBIDTOKEN, the returned SRBIDTOKEN will have a non-zero value in the first eight bytes.

SRBIDTOKEN is optional for PRIORITY=PREEMPT, PRIORITY=CLIENT, and PRIORITY=ENCLAVE.

,CLIENTSTOKEN=*clientstoken*
Specifies the space token (STOKEN) of the address space where the processor time used by the SRB is to be accumulated. The SRB also inherits the major priority of this address space. This parameter is a required input parameter for PRIORITY=CLIENT.

To code: Specify the name (RS-type), or address in register (2)-(12), of a required 64-bit parameter.

,ENCLAVETOKEN=*enclavetoken*
Specifies the enclave token representing the group of SRB routines. The enclave token must be obtained prior to scheduling the SRB.

To code: Specify the name (RS-type), or address in register (2)-(12), of an 8-character input parameter. ENCLAVETOKEN=*enclavetoken* is required for PRIORITY=ENCLAVE.

,FEATURE=NONE
,FEATURE=CRYPTO
Optional parameter that specifies affinity to specific processors.

NONE
Specifies that there is no affinity to specific processors.

CRYPTO
Specifies that the SRB routine must run on a processor that has an Integrated Cryptographic Feature (ICRF) associated with it. When you specify this parameter, the system assigns the correct processor affinity for the SRB routine. Use FEATURE=CRYPTO only for SRB routines whose exclusive purpose is to encrypt or decrypt data.

Default: NONE

LLOCK=NO
LLOCK=YES
Specifies whether the SRB is to receive control with the LOCAL lock held. The LOCAL lock is the lock of the home address space.

Default: NO

,FRRADDR=NOFRR
,FRRADDR=*frraddr*
Specifies the name (RS-type), or address in register (2)-(12), of an optional 4 byte input that contains the address of the Functional Recovery Routine (FRR) that is to be established prior to the SRB routine receiving control. The low bit of this address should not be set on. If it is set on, that bit will not be treated as part of the FRR address, but will be treated as indicating SDWALOC31=YES and will override the specification, or default, of SDWALOC31=NO.

The FRR receives control in supervisor state, PSW key 0, primary ASC mode, 31-bit addressing mode, holding the same locks the SRB routine held at the time of error. The FRR receives control with the same PASID, SASID, and HASID as the SRB routine had on entry.

If you specify LLOCK=YES, then the FRR should release the LOCAL lock prior to the completion of its processing.

Default: NOFRR. The SRB routine will receive control without its own FRR.

To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 4-byte input parameter.

SDWALOC31=NO
SDWALOC31=YES
Specifies whether the FRR specified by FRRADDR can tolerate an SDWA in 31-bit addressable storage. Considering that 31-bit storage is less likely to be constrained than 24-bit storage and RTM skips FRRs for which it can not obtain an SDWA, use SDWALOC31=YES whenever possible. SDWALOC31 is valid only for FRRADDR.

Default: NO

,KEYVALUE=INVOKERKEY
,KEYVALUE=*keyvalue*
Specifies the name or address of an optional 8-bit input. Bits 0-3 contain the PSW key in which the SRB is to receive control. Bits 4-7 are ignored. For example, the byte required to specify PSW key 7 contains the value X'70', and the byte required to specify PSW key 11 contains the value X'BO'.

Default: INVOKERKEY

If INVOKERKEY is not specified the SRB routine receives control with the PSW key of the invoker of the IEAMSCHD macro.

To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 8-bit input parameter.

RMTRADDR=NORMTR
RMTRADDR=*rmtraddr*
Specifies the address of an SRB resource manager termination routine (RMTR). RMTRs are responsible for cleaning up resources on behalf of an SRB routine that has been purged by the PURGEDQ service before the SRB is first dispatched.

The RMTR must reside in the MVS common area because the address space where the RMTR will get control is unpredictable at the time of the invocation of the IEAMSCHD macro. It is called from the PURGEDQ service and will receive control in task mode, supervisor state, PSW key 0, primary ASC mode, and 31-bit AMODE. If bit 31 of the RMTRADDR is one, control is received with the local lock held and control can return to the PURGEDQ service with or without the local lock held, but must not hold any other locks upon return. If bit 31 of the RMTRADDR is zero, control is received with no locks held and control must be returned to the PURGEDQ service with no locks held. Bit 31 of RMTRADDR is treated as zero when determining the RMTR address.

Default: NORMTR

To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 4-byte input parameter.

,PARM=ZERO
,PARM=*parm*
Specifies input to be loaded into register 1 when the SRB routine receives control.

Default: ZERO

To code: Specify the name (RS-type), or address in register (2)-(12), of a fullword input parameter.

,PURGESTOKEN=NOPSTOKEN
,PURGESTOKEN=*purgestoken*
Specifies the space token of an address space to be associated with this SRB routine. During memory termination, all SRB routines that are scheduled into the address space and have not received control are purged and control will be given to each SRB routine's RMTR.

The address space represented by the *purgestoken* does not have to be the same as the address space where the SRB routine will be dispatched.

Default: NOPSTOKEN

To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 64-bit input parameter.

,PTCBADDR=NOPTCB
,PTCBADDR=*ptcbaddr*
Specifies the address of the TCB to be related to the SRB routine. When a SYNCH=NO SRB routine scheduled with a related task terminates abnormally and the FRR for the SRB routine does not exist or percolates, the error is percolated to the recovery routine of the related task. This is known as SRB-to-task percolation.

If you specify PTCBADDR, then you must specify PURGESTOKEN.

Default: NOPTCB

To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 4-byte input parameter.

,FLAGS=NO_FLAGS
,FLAGS=*flags*
Specifies an optional 1-byte input/output field that provides information about the scheduling of the SRB. The caller must zero this byte before invoking IEAMSCHD.

- When bit 7 (the x'01' bit) is on, the SRB has been successfully scheduled.
- When bit 7 is off, but bit 6 (the x'02' bit) is on, the system had just begun the final part of the scheduling of the SRB, but that did not complete successfully; it is not known whether the SRB will or will not get control.
- When both bits 6 and 7 are off, the SRB was not successfully scheduled. This is an extremely rare circumstance.

These bit definitions are true whether control returns normally to the caller or whether control passes to the caller's recovery. The byte must be in disabled-reference or page-fixed storage.

Default: NO_FLAGS

To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 1-byte input output parameter.

,SYNCH=NO
,SYNCH=YES
Specifies whether the caller's work unit is to be suspended until the scheduled SRB completes, is purged, or ends abnormally:

SYNCH=NO
The SRB is to be scheduled but not synchronized with the caller's work unit.

SYNCH=YES
The SRB is to be scheduled and synchronized with the caller's work unit; the caller's work unit is suspended until the SRB completes, is purged, or ends abnormally. No IRBs are scheduled to run under the caller's work unit while it is suspended waiting for the SRB to complete.

Default: NO

Note: When a task invokes IEAMSCHD with SYNCH=YES, SRB to Task percolation is disabled because the task can use the SynchCompAddr, SynchCodeAddr, and SynchRsnAddr parameters to be notified of SRB ABENDs.

,SYNCHCOMPADDR=NOVALUE
,SYNCHCOMPADDR=*compaddr*
When you specify SYNCH=YES, you can specify this optional parameter, which contains one of the following completion codes when the caller's work unit resumes:

Code	Meaning
0	SRB completed successfully.
8	SRB ended abnormally; there is an associated reason code.
12	SRB ended abnormally; there is no associated reason code.
16	PURGEDQ processing purged the SRB.
20	SRB state is undetermined. It was dispatched but did not complete. A probable cause is address space termination or an error in the dynamic address translation (DAT) process.
24	SRB was not scheduled; SYNCHCODEADDR contains the return code from the SUSPEND service.
28	SRB was not scheduled; SYNCHCODEADDR contains the abend code from the SUSPEND service.

Default: NOVALUE

To code: Specify the name (RS-type) of an optional 4-byte input area that contains the address of the fullword that is to hold the data to be returned. When you specify this parameter, you must also specify SYNCHCODEADDR and SYNCHRSNADDR, which can provide additional information about the completion code.

,SYNCHCODEADDR=*codeaddr*
When the caller's work unit resumes, contains information associated with the completion code returned through SYNCHCOMPADDR. The completion codes and the associated information are:

Code	SYNCHCODEADDR Contents
0	Contents of GPR 15 when the SRB completed.
8	Abend code in the same format as field SDWAABCC in the SDWA.
12	Abend code in the same format as field SDWAABCC in the SDWA.
16	X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.

20
X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.

24
Return code from the SUSPEND service. The SRB was not scheduled because this work unit could not be successfully suspended.

28
Abend code from the SUSPEND service. The SRB was not scheduled because this work unit could not be successfully suspended.

For example, if SYNCHCOMPADDR contains a completion code of 8, then SYNCHCODEADDR contains an abend code. (If the scheduled SRB exits with the TCTL macro, SYNCHCODEADDR does not contain meaningful data; its contents are unpredictable.)

To code: Specify the name (RS-type) of an optional 4-byte input area that contains the address of the fullword that is to hold the data to be returned.

,SYNCHRSNADDR=*rsnaddr*
When the caller's work unit resumes, contains additional information associated with the completion code returned through SYNCHCOMPADDR and the information returned through SYNCHCODEADDR. The completion codes and the associated information are:

Code
SYNCHRSNADDR Contents

0
Contents of GPR 0 when the SRB completed.

8
Reason code associated with an abend code.

12
X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.

16
X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.

20
X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.

24
X'FFFFFFFF' (-1), indicating that there is no meaningful value to return.

28
Reason code associated with the abend code issued during an unsuccessful attempt to suspend the current work unit.

For example, if SYNCHCOMPADDR contains a completion code of 8, then SYNCHCODEADDR contains an abend code, and SYNCHRSNADDR contains the reason code associated with the abend code. (If the scheduled SRB exits with the TCTL macro, SYNCHCODEADDR and SYNCHRSNADDR do not contain meaningful data; the contents of both are unpredictable.)

To code: Specify the name (RS-type) of an optional 4-byte input area that contains the address of the fullword that is to hold the data to be returned.

,TRANSFER=NO
,TRANSFER=YES
An optional input parameter for a task scheduler of an SRB to specify whether the SRB can "take over" the current dispatch from the task (for instance, if the SRB work is considered to have higher importance than the task).

TRANSFER=NO
Specifies that the SRB is to be dispatched normally.

TRANSFER=YES
Specifies that an attempt is to be made to transfer control to the SRB if the requestor is running enabled and in task mode. If the transfer is successful, the requestor will be re-dispatched immediately.

TRANSFER=YES is ignored for the following cases:

- A SYNCH=YES request.
- Running in SRB mode.
- The caller holds any locks.

Default: NO

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
,PLISTVER=2
,PLISTVER=3
,PLISTVER=4
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports only the following parameters:
  - CLIENTSTOKEN
  - ENCLAVETOKEN
  - ENV
  - EPADDR
  - FEATURE
  - FRRADDR
  - KEYVALUE
  - LLOCK
  - MINORPRIORITY
  - PARM
  - PLISTVER
  - PRIORITY
  - PTCBADDR
  - PURGESTOKEN
  - RMTRADDR

  - TARGETSTOKEN
  - TRANSFER
- **1**, supports the following parameters, and parameters from version 0:
  - SYNCH
  - SYNCHCODEADDR
  - SYNCHCOMPADDR
  - SYNCHRSNADDR
- **2**, supports version 2 parameters (of which there are currently none documented), and parameters from any lower versions.
- **3**, supports the following parameter, and parameters from any lower versions:
  - SRBIDTOKEN
- **4**, supports the following parameter, and parameters from any lower versions:
  - FLAGS

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, 3, or 4

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

IEAMSCHD might abnormally end with system completion code AC7. See *z/OS MVS System Codes* for an explanation and programmer responses for this code.

Return codes

When the IEAMSCHD macro returns control to your program, GPR 15 contains a return code.

Table 50. Return Codes for the IEAMSCHD Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: Successful completion.
04	**Meaning**: Warning. The enclave token is not valid. The enclave token specified on the ENCLAVETOKEN parameter has been reused for a new enclave. The SRB was not scheduled.
08	**Meaning**: Program error. The client STOKEN address space has failed. The SRB was not scheduled.
0C	**Meaning**: Program error. The purge STOKEN address space has failed. The SRB was not scheduled.
10	**Meaning**: Program error. The target STOKEN address space has failed. The SRB was not scheduled.
1C	**Meaning**: Program error. A SYNCH=YES SRB was not scheduled or did not complete successfully. This is set only if the SYNCHCOMPADDR parameter is used with an operand that is not NOVALUE and does not indicate that location 0 is to be where data is returned. The values returned on SYNCHCOMPADDR, SYNCHCODEADDR, and SYNCHRSNADDR contain additional information.

Examples

Example 1

Schedule a local SRB routine to the current home address space. The SRB routine will be entered in the same key as the scheduling program.

```
*
SCHED_SRB_RTN  EQU      *
               IEAMSCHD EPADDR=EP_ADDR,ENV=HOME,PRIORITY=LOCAL
    .
    .
    .
EP_ADDR        DC   A(SRB_ROUTINE)   Address of Entry Point for SRB
*
```

Example 2

Schedule an SRB routine to the current primary address space which has the same addressability and cross memory environment as the scheduling program and has a GLOBAL priority. The SRB routine is to receive control in PSW Key 0.

```
*
SCHED_SRB_RTN  EQU *
               IEAMSCHD EPADDR=EP_ADDR,ENV=FULLXM,           X
                        PRIORITY=GLOBAL,                     X
                        KEYVALUE=PSW_KEY_0
    .
    .
    .
EP_ADDR        DC   A(SRB_ROUTINE)   Address of Entry Point for SRB
PSW_KEY_0      DC   X'00'            PSW Key 0
*
```

Example 3

Schedule an SRB routine at a priority that is the lowest in the enclave identified by the token in ENCLAVE_TOKEN. The SRB routine is to receive control in the current home address space with an FRR

established and holding the local lock of the current home address space. It is to run in the current home address space and is to run in key 2. The SRB routine has a resource manager termination routine whose entry point address is in RMTR_ADDR. The current task's recovery is to receive control should the SRB routine's recovery percolate and the SRB routine should be purged if the current task terminates. This example assumes that ENCLAVE_TOKEN and PURGE_STOKEN were previously initialized.

```
*
SCHED_SRB_RTN  EQU *
               USING  PSA,0          Base Prefixed Save Area
*
               IEAMSCHD  EPADDR=EP_ADDR,FRRADDR=FRR_ADDR,    X
               KEYVALUE=PSW_KEY_2,PRIORITY=ENCLAVE,          X
               ENCLAVETOKEN=ENCLAVE_TOKEN,                   X
               MINORPRIORITY=MINOR_PRIORITY,                 X
               RMTRADDR=RMTR_ADDR,                           X
               PURGESTOKEN=PURGE_STOKEN,                     X
               PTCBADDR=PSATOLD,                             X
               LLOCK=YES,ENV=HOME
*
    .
    .
    .
ENCLAVE_TOKEN  DS    D               Enclave Token
PURGE_STOKEN   DS    D               Purge-STOKEN
EP_ADDR        DC    A(SRB_ROUTINE)  SRB Entry Point Address
FRR_ADDR       DC    A(FRR_ROUTINE)  Address of FRR Routine
RMTR_ADDR      DC    A(RMTR_ROUTINE) RMTR Entry Point Address
MINOR_PRIORITY DC    X'00'          Lowest Priority in Enclave
PSW_KEY_2      DC    X'20'           PSW Key 2
               TITLE 'PSA -- Prefix Save Area'
               IHAPSA
```

Example 4

Schedule a LOCAL SRB routine into the address space whose STOKEN is stored in THEIR_STOKEN. This example assumes that THEIR_TOKEN was previously initialized.

```
*
SCHED_SRB_RTN  EQU *
               IEAMSCHD  EPADDR=EP_ADDR,ENV=STOKEN,          X
               TARGETSTOKEN=THEIR_STOKEN,                    X
               PRIORITY=LOCAL
*
    .
    .
    .
THEIR_STOKEN   DS    D               Space Token
EP_ADDR        DC    A(SRB_ROUTINE)  SRB Entry Point Address
```

Note that in this example, the SRB routine is running in a different address space from the scheduling code. To run an SRB routine in a different address space from the scheduling code, the SRB must be either in a different program that is accessible from the target address space, or in the common storage together with the scheduling code.

Example 5

Schedule a preemptable SRB routine into the current home address space with a minor priority that is just below the current task's dispatching priority.

```
*
SCHED_SRB_RTN  EQU *
*
               EXTRACT TCB_PRIORITY,'S',FIELDS=(PRI)
*
               SLR        3,3             Clear register
               IC         3,DSP_PRIORITY  Get Dispatching Priority
               S          3,=F'1'    Lower priority by 1
               BP         SAVE_MINOR_PRIORITY
               SLR        3,3        If tasks priority already lowest
*                                     set minor priority to zero.
SAVE_MINOR_PRIORITY EQU *
               STC        3,MINOR_PRIORITY  Save Minor Priority
```

```
               IEAMSCHD   EPADDR=EP_ADDR                      X
                          PRIORITY=PREEMPT,ENV=HOME,          X
                          MINORPRIORITY=MINOR_PRIORITY
*
    .
    .
EP_ADDR        DC    A(SRB_ROUTINE)  Address of Entry Point for SRB
TCB_PRIORITY   DS    0F              Priority Field
               DS    H               Place holder
DSP_PRIORITY   DS    B               Current Dispatching Priority
MINOR_PRIORITY DS    B               Minor Priority for SRB Routine
```

Example 6

Schedule an SRB routine into the home address space, passing it the parameter list pointed to by PARM_ADDR, and give the SRB routine affinity to online processors with the Integrated Cryptographic Feature installed. The SRB routine is to inherit the current work unit's major and minor priorities.

```
*
SCHED_SRB_RTN  EQU *
*
               IEAMSCHD   EPADDR=EP_ADDR,PARM=PARM_ADDR,     X
               FEATURE=CRYPTO,ENV=HOME,PRIORITY=CURRENT
*
    .
    .
    .
EP_ADDR        DC    A(SRB_ROUTINE)  SRB Entry Point Address
PARM_ADDR      DC    A(PARM_LIST)    Pointer to parameter list
```

Example 7

Schedule a synchronous LOCAL SRB routine into the address space whose STOKEN is stored in THEIR_STOKEN. The invoker of IEAMSCHD will be suspended until the SRB routine completes, abends, or is purged. This example assumes that THEIR_TOKEN was previously initialized.

```
*
SCHED_SRB_RTN  EQU *
               IEAMSCHD  EPADDR=EP_ADDR,ENV=STOKEN,          X
               TARGETSTOKEN=THEIR_STOKEN,PRIORITY=LOCAL,     X
               SYNCH=YES,SYNCHCOMPADDR=COMPCODE,             X
               SYNCHCODEADDR=ABENDCODE,SYNCHRSNADDR=REASONCODE
*
    .
    .
    .
THEIR_STOKEN   DS    D               Space Token
EP_ADDR        DC    A(SRB_ROUTINE)  SRB Entry Point Address
COMPCODE       DS    F
ABENDCODE      DS    F
REASONCODE     DS    F
```

Note that in this example, the SRB routine is running in a different address space from the scheduling code. To run an SRB routine in a different address space from the scheduling code, the SRB routine must be either in a different program that is accessible from the target address space, or in the common storage together with the scheduling code.

Chapter 44. IEAMSXMP — Safe cross-memory post

Description

The IEAMSXMP macro provides a safe cross-memory (XM) post protocol (in contrast with the POST macro for which cross-memory POST is safe only in certain cases). The safe XM post protocol

- Schedules an SRB with STOKEN to the target space, passing information to be used to validate the target, such as a TCB Token (TTOKEN) identifying a target TCB.
- The target SRB validates that the target TCB is valid, avoiding the POST if it is not
- If the target TCB is valid, uses branch-entry non-cross-memory POST to POST the ECB.

Environment

The requirements for the caller are:

Environmental Factor	Requirement
Minimum authorization:	Supervisor state. Any PSW key.
Dispatchable unit mode:	Task or SRB.
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	31- or 64-bit. If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	The caller can hold any LOCAL lock, the CMS lock, or the CPU lock but is not required to hold any.
Control parameters:	Control parameters must be in the primary address space. Control parameters above 2 GB are allowed for AMODE 64 callers only. The WorkArea must be below 2 GB.

Programming requirements

Make sure that field ECVTSXMP is nonzero before using this service. You can assume that to be the case for any z/OS release after z/OS V2R2.

Include the IEAASM macro to get equate symbols for return and reason codes.

Restrictions

None.

Input register information

Before issuing the IEAMSXMP macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IEAMSXMP macro, the caller does not have to place any information into any access register (AR).

Output register information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code, when register 15 is not 0; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

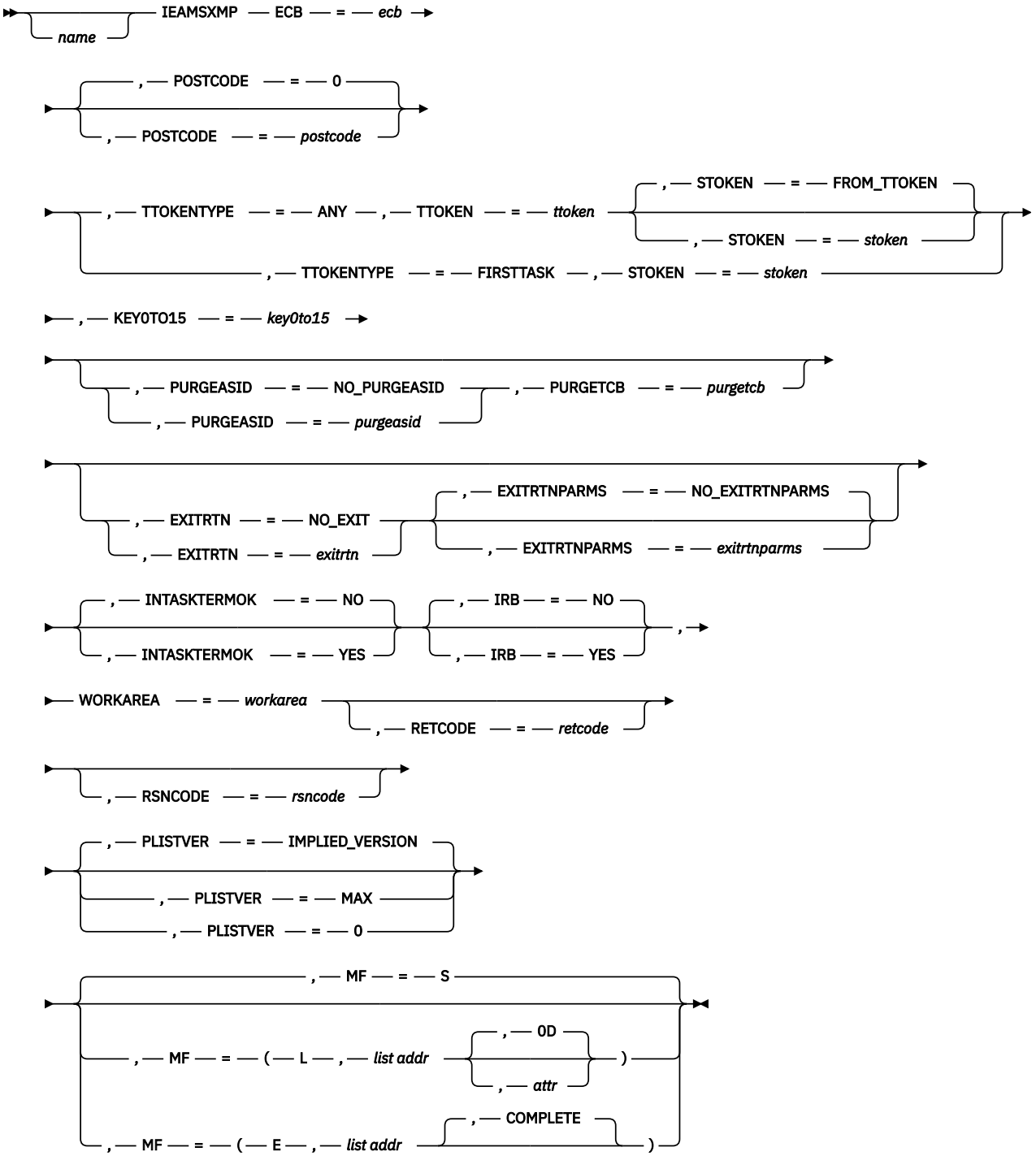
Some callers depend on register contents to remain the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Main diagram



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1 that is the name on the IEAMSXMP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

ECB=*ecb*

A required input parameter, which is, the ECB to post. The storage for the ECB must be synchronized with the TTOKEN that is used. Unless posted, the ECB storage must be valid (not freed) until termination of the task that is represented by the TTOKEN.

If you are changing from using POST to using IEAMSXMP, you can be introducing an incompatible storage persistence characteristic of the ECB storage that was not enforced or documented before the usage of IEAMSXMP. If the ECB to be posted is part of a documented interface, you must document any new storage persistence requirement that was not previously documented.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,EXITRTN=*exitrtn*
,EXITRTN=NO_EXIT

An optional input parameter that contains the address of an exit routine in common storage, which will execute in the following cases:

- When IRB=NO is in effect (or when IRB=YES was specified but TSO authorized request processing is not active), and the TTOKEN is known to be valid, within the scheduled SRB before the POST is issued:
  - It will be in the target address space.
  - It will be in SRB mode and will have the local lock. The exit routine must not release the lock.
  - A system-set FRR will be in place.
  - It must preserve registers 2 - 14.
  - It can indicate if the POST should or should not proceed by a return code in register 15:
    - 0 indicates that the POST should proceed.
    - 8 indicates that the POST should not proceed.

    Do not use any other return code.
- When IRB=YES is in effect and TSO authorized request processing is active, and the TTOKEN is known to be valid, within the IRB before the POST is issued:
  - It will be in the target address space.
  - It will be in task mode and will have the local lock. The exit routine must not release the lock.
  - No FRR will be in place.
  - It must preserve registers 2 - 14.
  - It can indicate if the POST should or should not proceed by a return code in register 15:
    - 0 indicates that the POST should proceed.
    - 8 indicates that the POST should not proceed.

    Do not use any other return code.
- When the scheduled SRB determines that the TTOKEN is not valid.
  - It will be in the target address space.
  - It will be in SRB mode and will have the local lock. The exit routine must not release the lock.
  - A system-set FRR will be in place.
  - It must preserve registers 2 - 14.
  - It might, for example, free the ECB if that ECB is known still to exist.
- If the scheduled SRB is purgeDQ because of termination of the ASID (identified by PurgeASID) or the task (identified by PurgeTCB) or because of a user-issued purgeDQ.
  - It can be in any address space.
  - It will be in task mode.

- A system-set FRR will be in place.
- It will not hold any locks.
- It must preserve registers 2 - 14.

The exit routine is responsible for its own recovery. If the exit routine does not return to the system, the ECB can not be posted.

The exit routine gets control in key 0 supervisor state, primary ASC mode, primary = home = secondary, AMODE 31

Register contents on entry:

0	**Type of call**
0	Valid TTOKEN, IRB=NO
1	Valid TTOKEN, IRB=YES
2	TTOKEN is not valid
3	PurgeDQ
1	Address of a copy of the 16 bytes of data provided by the EXITRTNPARMS parameter.
2-13	Do not contain information for use by the exit routine
14	Return address
15	Address of exit routine

The default is NO_EXIT.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,EXITRTNPARMS=*exitrtnparms*
,EXITRTNPARMS=NO_EXITRTNPARMS
When EXITRTN=*exitrtn* is specified, an optional input parameter, 16 bytes of data to be passed to the exit routine. One use of this area could be to pass the address of a storage area that the exit routine can use. The default is NO_EXITRTNPARMS.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,INTASKTERMOK=NO
,INTASKTERMOK=YES
An optional parameter that indicates if the case of "within task termination" is to be considered as OK when validating the task associated with the ECB to be posted. The default is INTASKTERMOK=NO.

,INTASKTERMOK=NO
Indicates that within task termination is not considered OK.

,INTASKTERMOK=YES
Indicates that within task termination is considered OK. When this option is used, the ECB storage must not be freed except implicitly by the system (such as upon task termination) unless it is known that there will be no subsequent attempt to POST that ECB (for example, it has already been POSTed or processing has not gotten far enough that it would ever be POSTed).

,IRB=NO
,IRB=YES
An optional parameter that indicates if special IRB-related processing is to be done. The default is IRB=NO.

,IRB=NO
Indicates that no special IRB-related processing is to be done.

,IRB=YES
Indicates that an IRB will be scheduled out of the scheduled SRB when TSO authorized request processing is active. TSO allows authorized code to manipulate key 8 storage and attempts to

maintain integrity by marking all unauthorized tasks as non-dispatchable. Scheduling the IRB to do the post for this case ensures that the post does not affect the processing of authorized code. The IRB will run under the task that is identified by the TTOKEN keyword. When TCB authorized request processing is not active, IRB=NO processing will be done.

If a user key (8-15) ECB is to be posted that your product or component does not own for a target address space that your product or component does not own, the IRB=YES option must be used to ensure that the post operation is secure.

,KEY0TO15=*key0to15*

A required input parameter that is the key with which the POST is to be done. Must be in the range 0-15 (not X'00' - X'F0').

To code: Specify the RS-type address, or address in register (2) - (12), of a 1-byte field.

,MF=S

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

,MF=(L,*list addr* | *attr* | 0D)

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

,MF=(E,*list addr* | COMPLETE)

MF=E is an optional input parameter that specifies the macro form. This parameter specifies the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area that is defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr

The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr

An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE

Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- IMPLIED_VERSION, which is the lowest version that allows all parameters that are specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- MAX, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- **IMPLIED_VERSION**
- **MAX**
- A decimal value of **0**

,POSTCODE=*postcode*
,POSTCODE=0
An optional input parameter, which is the POST code to use. The POST code is intended to be in the range 0 - 2**24-1. The default is 0.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,PURGEASID=*purgeasid*
,PURGEASID=NO_PURGEASID
An optional input parameter that is the purge ASID for the scheduled SRB. If used, and the POST abends, the task in this address space that is identified by the purgeTCB keyword will be abended. If not specified, or if a value of 0 is provided, the target address space's ASID is used. The default is NO_PURGEASID.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,PURGETCB=*purgetcb*
When PURGEASID=*purgeasid* is specified, a required input parameter that is the purge TCB for the scheduled SRB. If a non-0 value is used and the scheduled SRB abends, the purge TCB task will be abended.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,STOKEN=*stoken*
,STOKEN=FROM_TTOKEN
When TTOKENTYPE=ANY is specified, an optional input parameter that is the STOKEN for the target address space. Use this if you are providing a TTOKEN of zeros. You can use this, but do not need to, if providing a TTOKEN obtained through the TCBTOKEN service or from field STCBTTKN. You might have gotten the STOKEN from field ASSBSTKN while running in the address space or through LOCASCB STOKEN=. You must provide a valid STOKEN and not use STOKEN=FROM_TTOKEN if using a TTOKEN of zeros. If you provide a valid STOKEN with a valid TTOKEN, both must refer to the same address space. The default is FROM_TTOKEN.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,STOKEN=*stoken*
When TTOKENTYPE=FIRSTTASK is specified, a required input parameter that is the STOKEN for the target address space. You might have gotten this from field ASSBSTKN while running in the address space or through LOCASCB STOKEN=.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,TTOKEN=*ttoken*
When TTOKENTYPE=ANY is specified, a required input parameter that is the TTOKEN for the task to be validated. It might have been fetched using the TCBTOKEN service. A value of zeros is treated the same as TTOKENTYPE=FIRSTTASK and has the same requirements as TTOKENTYPE=FIRSTTASK. The target space STOKEN will be derived from the TTOKEN unless the STOKEN parameter is used and provides a nonzero STOKEN.

The task must be active in order for the IEAMSXMP operation to succeed. The rule is that the ECB must persist until it is posted or until the persistence requirement of the poster is met. In general, the TTOKEN to use can be determined as follows:

- If the target ECB is supplied by a task and it is expected that the target ECB can only be posted up until the time that task terminates, then use the current task's STCBTTKN for the TTOKEN.
- If the target ECB is supplied from a task that is part of the jobstep program task tree (the current task's TCBBITCB bit is on) and it is expected that the ECB can be posted up until the time the currently running job step ends, then use the TTOKEN in STCBTTKN for the task with TCB address in field ASCBXTCB.
- If it is expected that the target ECB can be posted up until the time that the first task in the address space terminates, then specify a TTOKEN of hexadecimal zeros or use the TTOKENTYPE=FIRSTTASK option. Provide the STOKEN.

The poster "owns" the rule and should document that rule for the ECB-related parameter of their interface. That rule might be one of the above choices or might be something else. For example, there might be one rule for a request from a task within the jobstep program task tree and a different rule for a request from a task not within that task tree (which would typically be a system task).

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,TTOKENTYPE=ANY
,TTOKENTYPE=FIRSTTASK
A required parameter that indicates the type of TCB Token (TTOKEN) to be used for validation

,TTOKENTYPE=ANY
That indicates any TTOKEN, not a specific type

,TTOKENTYPE=FIRSTTASK
that indicates to use the TTOKEN of the first task for the address space (this task will be the region control task for all address spaces other than *MASTER* which has no RCT) and its address can be found in ASXBFTCB. TTOKENTYPE=FIRSTTASK should usually be used only for a space that always terminates when the jobstep program task terminates, unlike an initiator space. It can also be used for compatibility purposes when changing from POST to IEAMSXMP when you have to consider the existing use cases. Use of TTOKENTYPE=FIRSTTASK requires use of STOKEN, and that STOKEN must have been fetched with serialization against address space termination or must have been captured while the target space is your home, primary, or secondary address space. Unless it has been posted, the ECB must not be freed (explicitly or by the system) prior to the termination of the address space's first task (which occurs only during memory termination). Thus the ECB storage must not be owned by a subtask of that first task (since the system frees storage owned by a task upon termination of that task).

,WORKAREA=*workarea*
A required input parameter, which is a 512-byte work area on a doubleword boundary that is below 2G, addressable in the primary address space, for use by the service. It can be in private storage. if the caller is not enabled for external and I/O interrupts, the work area must be page-fixed. If the caller is enabled for external and I/O interrupts, the work area need not be page-fixed. The work area must have a storage key that accommodates a store using the PSW key of the invoker of IEAMSXMP.

To code: Specify the RS-type address, or address in register (2) - (12), of a 512-character field.

ABEND codes

None.

Return and reason codes

When the IEAMSXMP macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro IEAASM provides equate symbols for the return and reason codes. It also provides equate IEAMSXMPRsnCodeMask. That equate can be used to create a word that should be ANDed with the reason code to isolate the non component-diagnostic portion of the reason code before doing a comparison.

The following table identifies the hexadecimal return and reason codes and the equate symbol that is associated with each reason code. IBM support personnel can request the entire reason code, including the *xxxx* value.

Table 51. Return and reason codes for the IEAMSXMP macro

Return code	Reason code	Equate symbol, meaning and action
0	–	**Equate symbol**: IEAMSXMPRc_OK **Meaning**: The POSTing SRB has been successfully scheduled. **Action**: None required
4	–	**Equate symbol**: IEAMSXMPRc_Warn **Meaning**: Warning **Action**: Refer to the action under the individual reason code.
4	*xxxx*0401	**Equate symbol**: IEAMSXMPRsn_TargetNotValid **Meaning**: The system found that the target address space was not valid. The ECB will not be posted. **Action**: If it is expected that the target address space could be not valid, then no action is required but you might be able to free the storage for the ECB. If it is not expected, then provide a correct target address space.
8	–	**Equate symbol**: IEAMSXMPRc_InvParm **Meaning**: The IEAMSXMP invocation specified parameters that are not valid **Action**: Refer to the action under the individual reason code.
8	*xxxx*0801	**Equate symbol**: IEAMSXMPRsn_BadParmlist **Meaning**: Error accessing the parameter list **Action**: Check for possible storage overlay
0C	–	**Equate symbol**: IEAMSXMPRc_Env **Meaning**: Environmental Error **Action**: None - no such reason codes currently exist.
10	–	**Equate symbol**: IEAMSXMPRc_CompError **Meaning**: Unexpected failure. **Action**: Report the associated reason code to the system programmer to contact IBM Service.

Table 51. Return and reason codes for the IEAMSXMP macro (continued)		
Return code	**Reason code**	**Equate symbol, meaning and action**
10	*xxxx*1001	**Equate symbol**: IEAMSXMPRsn_CompError **Meaning**: Unexpected failure. The state of the request is unpredictable. **Action**: Contact your system programmer.

Example

Post the ECB providing the validation TTOKEN. The target address space will be determined from the TTOKEN.

The code is as follows:

```
* Code to avoid invoking IEAMSXMP if field
* ECVTSXMP is 0
 ...
* Code to put the address of the ECB into register n
 ...
* Invoke IEAMSXMP
         IEAMSXMP ECB=(n),POSTCODE=pc,                       *
               TTOKENTYPE=ANY,TTOKEN=tt,                     *
               KEY0TO15=0,                                   *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               WORKAREA=wa,MF=(E,SXMPL)
* Here you would place code to process the return and
* reason codes.
 ...
DYNAREA  DSECT
         DS    0D
wa       DS    CL512
pc       DS    F
tt       DS    CL16
LRETCODE DS    F
LRSNCODE DS    F
         IEAMSXMP MF=(L,SXMPL),PLISTVER=MAX
```

Chapter 45. IEANTCR — Create a name/token pair

Description

Call the IEANTCR service to create a name/token pair.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key. **Note:** Problem-state programs with PSW key 8 - 15 cannot create system-level pairs.
Dispatchable unit mode:	Task or SRB **Note:** SRB-mode callers cannot create a task-level pair.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	The parameter list and all parameters must reside in the caller's primary address space.

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

```
* Name/Token Level Constants
*
IEANT_TASK_LEVEL             EQU     1
IEANT_HOME_LEVEL             EQU     2
IEANT_PRIMARY_LEVEL          EQU     3
IEANT_SYSTEM_LEVEL           EQU     4
IEANT_TASKAUTH_LEVEL         EQU     11
IEANT_HOMEAUTH_LEVEL         EQU     12
IEANT_PRIMARYAUTH_LEVEL      EQU     13
*
* Name/Token Persistence Constants
*
IEANT_NOPERSIST              EQU     0
IEANT_PERSIST                EQU     1
IEANT_NOCHECKPOINT           EQU     0
IEANT_CHECKPOINTOK           EQU     2
*
* Name/Token Return Code Constants
*
IEANT_OK                     EQU     0
IEANT_DUP_NAME               EQU     4
IEANT_NOT_FOUND              EQU     4
IEANT_24BITMODE              EQU     8
IEANT_NOT_AUTH               EQU     16
IEANT_SRB_MODE               EQU     20
IEANT_LOCK_HELD              EQU     24
IEANT_LEVEL_INVALID          EQU     28
```

```
IEANT_NAME_INVALID          EQU     32
IEANT_PERSIST_INVALID       EQU     36
IEANT_AR_INVALID            EQU     40
IEANT_UNEXPECTED_ERR        EQU     64
```

Restrictions

Do not use the IEANTCR callable service in a RESMGR resource manager routine unless one of the following is true:

- The name/token pair is a system-level persistent name/token pair.
- The resource manager is running for a daughter task of the task that owns the name/token pair.
- The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information

Before issuing the IEANTCR callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL IEANTCR	,(level ,user_name ,user_token ,persist_option ,return_code)

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEANTCR:

```
1.   LOAD EP=IEANTCR
     Save the entry point address
     (...)
     Put the saved entry point address into R15
     CALL (15),(...)

2.   L     15,X'10'
     L     15,X'220'(15,0)
     L     15,X'14'(15,0)
     L     15,X'04'(15,0)
     CALL  (15),(...)
```

This second technique requires AMODE=31, and, before the CALL is issued, verification that the IEANTCR service is supported by the system (in the CVT, both the CVTOSEXT and the CVTOS390 bits are set on).

Parameters

The parameters are explained as follows:

level
Specifies a fullword that contains an integer indicating the level of the name/token pair:

1
Task

2
Home address space

3
Primary address space

4
System.

,user_name
Specifies the 16-byte area containing the name of the name/token pair that the user creates. The bytes of the name may have any value. The name may contain blanks, integers, or addresses.

Names must be unique within a level. Here are some examples.

- Two task-level name/token pairs owned by the same task cannot have the same name. However, two task-level name/token pairs owned by different tasks can have the same name.
- Two home-address-space-level name/token pairs in the same address space cannot have the same name. However, two home-address-space-level name/token pairs in different address spaces can have the same name.

Because of these unique requirements you must avoid using the same names that IBM uses for name/token pairs. Do not use the following names:

- Names that begin with A through I
- Names that begin with X'00'.

,user_token
Specifies the 16-byte area containing the token of the name/token pair that the user creates.

,persist_option
Specifies a fullword that contains an integer indicating if a system-level name/token pair should persist after the creating address space's job step task terminates or if Checkpoint/Restart can be issued if the program has this task-level name/token pair. If a program has non-task-level name/token pairs or has task-level name/token pairs that did not specify IEANT_CHECKPOINTOK, the program cannot take a checkpoint.

- 0 - system-level pair will not persist and checkpoint is not permitted.
- 1 - system-level pair will persist.
- 2 - checkpoint is permitted.

Note: Only system-level name/token pairs can persist after the creating task terminates. Only task-level name/token pairs can permit checkpoint.You must specify 0 for all other levels.

,return_code
Specifies a fullword to contain the return code from the IEANTCR service.

ABEND codes

The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See *z/OS MVS System Codes* for an explanation and responses for these codes.

Return and reason codes

When IEANTCR returns control to your program, GPR 15 and *return_code* contain a return code. The following table identifies return codes in hexadecimal and decimal, tells what each means, and recommends an action that you need to take:

Table 52. Return Codes for the IEANTCR Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The operation was successful. **Action**: None.
04	**Meaning**: The *user_name* specified already exists. **Action**: Choose a different *user_name*.
08	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 31-bit addressing mode.
10	**Meaning**: An unauthorized caller attempted to create a system-level name/token pair. **Action**: Check which level of name/token pair you are creating.
14	**Meaning**: An SRB-mode caller attempted to create a task-level name/token pair. **Action**: Change your program to task mode or use a different level.
18	**Meaning**: The caller held locks. **Action**: Release all locks before issuing IEANTCR.
1C	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid options are 1, 2, 3, or 4.
20	**Meaning**: The caller specified an incorrect *user_name*. **Action**: Respecify the correct *user_name*.

Table 52. Return Codes for the IEANTCR Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
24	**Meaning**: The caller specified an incorrect *persist_option*. **Action**: • For system-level name/token pairs, you must specify zero or one for the *persist_option*. • For task-level name/token pairs, you must specify zero or two for the *persist_option*. • For home or primary address space level name/token pairs, you must specify zero for the *persist_option*.
28	**Meaning**: The caller was in AR ASC mode and AR1 was not zero. **Action**: Change your program to primary mode or set AR1 to zero.
40	**Meaning**: A system error occurred while handling the request. **Action**: Retry the request.

Example

Initialize the name/token fields, and create, retrieve, and delete a task-level name/token pair.

```
         TITLE 'NAME/TOKEN EXAMPLE PROGRAM'
NTIDSAMP CSECT
NTIDSAMP AMODE 31
NTIDSAMP RMODE ANY
         BAKR  R14,0                  Save calling programs
*                                     registers and return location
         LR    R12,R15                ESTABLISH BASE REG
         USING NTIDSAMP,R12
*********************************************************************
*  INITIALIZE THE NAME AND TOKEN FIELDS                             *
*********************************************************************
         MVC   NAME,=CL16'NTIDSAMP NAME   ' INITIALIZE NAME FIELD
         MVC   TOKEN,NAME             FOR EXAMPLE, MAKE TOKEN THE
*                                     SAME AS THE NAME
*********************************************************************
*  TASK LEVEL CREATE EXAMPLE                                        *
*********************************************************************
         CALL  IEANTCR,(LEVEL,NAME,TOKEN,PERSOPT,RETCODE)
*********************************************************************
         CLC   RETCODE,=F'0'          IS RETURN CODE 0?
         BNE   ABEND                  NO, GO ABEND
         EJECT
*********************************************************************
*  TASK LEVEL RETRIEVE EXAMPLE                                      *
*********************************************************************
         CALL  IEANTRT,(LEVEL,NAME,TOKEN,RETCODE)
*********************************************************************
         CLC   RETCODE,=F'0'          IS RETURN CODE 0?
         BNE   ABEND                  NO, GO ABEND
         EJECT
*********************************************************************
*  TASK LEVEL DELETE EXAMPLE                                        *
*********************************************************************
         CALL  IEANTDL,(LEVEL,NAME,RETCODE)
*********************************************************************
         CLC   RETCODE,=F'0'          IS RETURN CODE 0?
         BNE   ABEND                  NO, GO ABEND
         EJECT
         SLR   R15,R15                SET RETURN CODE OF ZERO
EXIT     PR                           RETURN TO CALLER
         EJECT
ABEND    ABEND X'BAD'                 ABEND IF NONZERO RETURN CODE
         EJECT
*********************************************************************
* NAME/TOKEN VARIABLE DECLARES
*********************************************************************
         IEANTASM
         EJECT
*********************************************************************
* Constants and data areas                                          *
*********************************************************************
LEVEL    DC    A(IEANT_TASK_LEVEL)     Task level
```

```
NAME     DS    CL16                    Name for name/token pair
TOKEN    DS    XL16                    Token for name/token pair
PERSOPT  DC    A(IEANT_NOPERSIST)      Persist option
RETCODE  DS    F                       Return code
*********************************************************************
*  EQUATES
*********************************************************************
R1       EQU   1
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END   NTIDSAMP
```

Chapter 46. IEANTDL — Delete a name/token pair

Description

Call the IEANTDL service to delete a name/token pair.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key **Note:** Problem-state programs with PSW key 8 - 15 cannot delete: • System-level pairs • Name/token pairs created by supervisor-state or PSW key 0-7 programs.
Dispatchable unit mode:	Task or SRB **Note:** SRB-mode callers cannot delete a task-level pair.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	The parameter list and all parameters must reside in the caller's primary address space.

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

```
* Name/Token Level Constants
*
IEANT_TASK_LEVEL             EQU     1
IEANT_HOME_LEVEL             EQU     2
IEANT_PRIMARY_LEVEL          EQU     3
IEANT_SYSTEM_LEVEL           EQU     4
IEANT_TASKAUTH_LEVEL         EQU     11
IEANT_HOMEAUTH_LEVEL         EQU     12
IEANT_PRIMARYAUTH_LEVEL      EQU     13
*
* Name/Token Persistence Constants
*
IEANT_NOPERSIST              EQU     0
IEANT_PERSIST                EQU     1
*
* Name/Token Return Code Constants
*
IEANT_OK                     EQU     0
IEANT_DUP_NAME               EQU     4
IEANT_NOT_FOUND              EQU     4
IEANT_24BITMODE              EQU     8
IEANT_NOT_AUTH               EQU     16
IEANT_SRB_MODE               EQU     20
```

```
IEANT_LOCK_HELD              EQU     24
IEANT_LEVEL_INVALID          EQU     28
IEANT_NAME_INVALID           EQU     32
IEANT_PERSIST_INVALID        EQU     36
IEANT_AR_INVALID             EQU     40
IEANT_UNEXPECTED_ERR         EQU     64
```

Restrictions

Do not use the IEANTDL callable service in a RESMGR resource manager routine unless one of the following is true:

- The name/token pair is a system-level persistent name/token pair.
- The resource manager is running for a daughter task of the task that owns the name/token pair.
- The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information

Before issuing the IEANTDL callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL IEANTDL	,(level ,user_name ,return_code)

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEANTDL:

```
1.   LOAD EP=IEANTDL
     Save the entry point address
     (...)
     Put the saved entry point address into R15
     CALL (15),(...)

2.   L     15,X'10'
     L     15,X'220'(15,0)
     L     15,X'14'(15,0)
     L     15,X'0C'(15,0)
     CALL  (15),(...)
```

This second technique requires AMODE=31, and, before the CALL is issued, verification that the IEANTDL service is supported by the system (in the CVT, both the CVTOSEXT and the CVTOS390 bits are set on).

Parameters

The parameters are explained as follows:

level
Specifies a fullword that contains an integer indicating the level of the name/token pair you wish to delete:

1
Task

2
Home address space

3
Primary address space

4
System.

,user_name
Specifies the 16-byte area containing the name of the name/token pair to be deleted.

,return_code
Specifies a fullword to contain the return code from the IEANTDL service.

ABEND codes

The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See *z/OS MVS System Codes* for an explanation and responses for these codes.

Return and reason codes

When IEANTDL returns control to your program, GPR 15 and *return_code* contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends an action that you need to take.

Table 53. Return Codes for the IEANTDL Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The operation was successful. **Action**: None.
04	**Meaning**: The request is rejected because the system could not find the requested name/token pair. **Action**: Check the *user_name* you specified.
08	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 31-bit addressing mode.
10	**Meaning**: An unauthorized caller attempted to delete a system-level pair or a name/token pair that was created by an authorized program. **Action**: Check which level of name/token pair you are deleting.
14	**Meaning**: An SRB-mode caller attempted to delete a task-level name/token pair. **Action**: Change the program to task mode or check the value you set for the level parameter.
18	**Meaning**: The caller held locks. **Action**: Release all locks before issuing IEANTDL.
1C	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid options are 1, 2, 3, or 4.
20	**Meaning**: The caller specified an incorrect *user_name*. **Action**: Respecify the correct *user_name*.
28	**Meaning**: The caller was in AR ASC mode and AR1 was not zero. **Action**: Change your program to primary mode or set AR1 to zero.
40	**Meaning**: A system error occurred while handling the request. **Action**: Retry the request.

Example

For a complete example of creating, retrieving, and deleting a task-level name/token pair, see the IEANTCR callable service.

Chapter 47. IEANTRT — Retrieve the token from a name/token pair

Description

Call the IEANTRT service to retrieve the token from a name/token pair.

The IEANTRT callable service can also be used to obtain the name of the logrec medium, either the name of the logrec data set or the name of the logrec log stream.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key
Dispatchable unit mode:	Task or SRB **Note:** SRB-mode callers cannot retrieve a task-level pair.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller can hold a local, CML, or CMS lock; however, no locks are required.
Control parameters:	The parameter list and all parameters must reside in the caller's primary address space.

Programming requirements

Before you use name/token services, you can optionally include macro IEANTASM to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

```
* Name/Token Level Constants
*
IEANT_TASK_LEVEL             EQU     1
IEANT_HOME_LEVEL             EQU     2
IEANT_PRIMARY_LEVEL          EQU     3
IEANT_SYSTEM_LEVEL           EQU     4
IEANT_TASKAUTH_LEVEL         EQU     11
IEANT_HOMEAUTH_LEVEL         EQU     12
IEANT_PRIMARYAUTH_LEVEL      EQU     13
*
* Name/Token Persistence Constants
*
IEANT_NOPERSIST              EQU     0
IEANT_PERSIST                EQU     1
*
* Name/Token Return Code Constants
*
IEANT_OK                     EQU     0
IEANT_DUP_NAME               EQU     4
IEANT_NOT_FOUND              EQU     4
IEANT_24BITMODE              EQU     8
IEANT_NOT_AUTH               EQU     16
IEANT_SRB_MODE               EQU     20
IEANT_LOCK_HELD              EQU     24
```

```
IEANT_LEVEL_INVALID          EQU     28
IEANT_NAME_INVALID           EQU     32
IEANT_PERSIST_INVALID        EQU     36
IEANT_AR_INVALID             EQU     40
IEANT_UNEXPECTED_ERR         EQU     64
```

To obtain the name of the logrec data set or the name of the logrec log stream, you can include the IFBNTASM macro, as well as the IEANTASM macro, in your program. See "Example 2" on page 582 for the list of definitions IFBNTASM provides.

Restrictions

- Do not use the IEANTRT callable service in a RESMGR resource manager routine unless one of the following is true:
  - The name/token pair is a system-level persistent name/token pair.
  - The resource manager is running for a daughter task of the task that owns the name/token pair.
  - The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.
- Do not call the IEANTRT callable service with *user_name* and *user_token* parameters being the same storage locations.

Input register information

Before issuing the IEANTRT callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL IEANTRT	,(level ,user_name ,user_token ,return_code)

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEANTRT:

```
1.   LOAD EP=IEANTRT
     Save the entry point address
     (...)
     Put the saved entry point address into R15
     CALL (15),(...)

2.   L     15,X'10'
     L     15,X'220'(15,0)
     L     15,X'14'(15,0)
     L     15,X'08'(15,0)
     CALL  (15),(...)
```

This second technique requires AMODE=31, and, before the CALL is issued, verification that the IEANTRT service is supported by the system (in the CVT, both the CVTOSEXT and the CVTOS390 bits are set on).

Parameters

The parameters are explained as follows:

level
Specifies a fullword that contains an integer indicating the level of the name/token pair from which you want to retrieve the token:

1
Task

2
Home address space

3
Primary address space

4
System

11
Task with authorization check

12
Home address space with authorization check

13
Primary address space with authorization check.

Note: Levels 11, 12, and 13 indicate that the IEANTRT callable service should determine if the name/token pair being retrieved was created by an authorized program.

,user_name
Specifies the 16-byte area containing the name of the requested name/token pair.

,user_token
Specifies the 16-byte area to contain the token of the requested name/token pair.

,return_code
Specifies a fullword to contain the return code from the IEANTRT service.

ABEND codes

None.

Return codes

When IEANTRT returns control to your program, GPR 15 and *return_code* contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends as action that you need to take.

Table 54. Return Codes for the IEANTRT Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The operation was successful. **Action**: None.
04	**Meaning**: The request is rejected because the system could not find the requested name/token pair. **Action**: Check the *user_name* you specified.
08	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 31-bit addressing mode.
10	**Meaning**: A request for a retrieval with authorization check attempted to retrieve a name/token pair created by an unauthorized caller. **Action**: If your program is authorized, you need to make sure that the name/token pair you are retrieving was created by another authorized program. You may choose to use the name/token pair if it was created by an unauthorized program, but doing so might cause data integrity problems.
14	**Meaning**: An SRB-mode caller attempted to retrieve a task-level name/token pair. **Action**: Check which level of name/token pair you are retrieving.
1C	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid options are 1, 2, 3, 4, 11, 12, or 13.
40	**Meaning**: A system error occurred while handling the request. **Action**: Retry the request.

Example 1

For a complete example of creating, retrieving, and deleting a task-level name/token pair, see the IEANTRT callable service.

Example 2

Following is an example of using Name/Token services to obtain the name of the logrec data set or logrec log stream. (Note that because the routine is not reentrant, module IEANTRT is first loaded and then called.) IEANTRT returns a token that contains a pointer to the name of the logrec data set or logrec log stream.

Before you use name/token services, you can optionally include macro IFBNTASM which provides the following definitions for use in your program:

```
* IFBNTASM Parameters

IFBNT_DSNLOGREC       DC     CL16'DSNLOGREC       '   System level
*                                 DSNLOGREC name
IFBNT_VERSION1        EQU    X'01'     First version of IFBNT_TOKEN
IFBNT_VERSION2        EQU    X'02'     Second version of IFBNT_TOKEN
IFBNT_LATEST_VERSION  EQU    X'02'     Latest version of IFBNT_TOKEN
*
IFBNT_TOKEN           DSECT  ,         Token area
IFBNT_LOGREC_NAME_PTR DS     A         Address of the LOGREC data
*                                      set name area
IFBNT_VERSION         DS     X         Version of IFBNT_LOGREC
IFBNT_RESV1           DS     X         Reserved for IBM
IFBNT_LENGTH          DS     XL2       Length of IFBNT_LOGREC area
IFBNT_RESV2           DS     CL8       Reserved for IBM
*
IFBNT_LOGREC          DSECT  ,         Pointed to by
*                                      IFBNT_LOGREC_NAME_PTR
IFBNT_LOGREC_NAME     DS     CL44      LOGREC data set name or
*                                      no data set name string (see
*                                      comments at end of mapping)
IFBNT_LOGREC_CURRENT  DS     XL1       Current Logrec recording
*                                      medium
IFBNT_LOGREC_PREVIOUS DS     XL1       Previous Logrec recording
*                                      medium
IFBNT_LOGREC_LOGSTREAM DS     CL26     Logrec log stream name,
*                                      only filled in when
*                                      IFBNT_USE_LOGSTREAM is
*                                      the current medium
IFBNT_LOGREC_LEN      EQU    *-IFBNT_LOGREC  Length of IFBNT_LOGREC
*
*********************************************************************
* The following values are used in the following fields:
*   IFBNT_LOGREC_CURRENT
*   IFBNT_LOGREC_PREVIOUS
*********************************************************************
IFBNT_USE_DATASET      EQU    X'01'     Logrec data set being used
IFBNT_USE_LOGSTREAM    EQU    X'02'     Logrec log stream being used
IFBNT_IGNORE_RECORDS   EQU    X'03'     Logrec recording is ignored
*
*********************************************************************
* If a Logrec data set was not defined during the IPL of the system
* then the following string will appear in field
* IFBNT_LOGREC_NAME = '...NO.LOGREC.DATA.SET.DEFINED...            '
*********************************************************************
```

IFBNT_TOKEN provides a DSECT to map the returned token area.

IFBNT_LOGREC_NAME_PTR contains the address of the logrec data set name.

IFBNT_LOGREC provides a DSECT to map the logrec recording medium.

IFBNT_LOGREC_NAME contains the name of the installation-defined logrec data set or no data set name, if the recording medium is other than a data set.

```
         TITLE 'DSNLOGREC Name/Token Retrieve Example Routine'
IFBNTXMP AMODE 31
IFBNTXMP RMODE ANY
IFBNTXMP CSECT
         BAKR  R14,0                  Save calling program's
*                                     registers and return location
         LR    R12,R15                Establish base ref
         USING IFBNTXMP,R12           Set addressability
         MODID BRANCH=YES
*********************************************************************
*  Initialize the NAME field
*********************************************************************
         MVC   NAME,IFBNT_DSNLOGREC   Request DSNLOGREC name
*********************************************************************
*  System level DSNLOGREC Retrieve example
*********************************************************************
         LOAD  EP=IEANTRT             Get address of IEANTRT routine
         LR    R15,R0                 Set address for Call
         CALL  (15),(LEVEL,NAME,TOKEN,RETCODE)
*
         LA    R15,IEANT_OK           Get successful return code value
```

```
         C     R15,RETCODE             Was TOKEN Returned?
         BNE   ABEND                   No, Go ABEND
         EJECT
*********************************************************************
*  Get the installation specified LOGREC data set name
*********************************************************************
         LA    R2,TOKEN                Set pointer to TOKEN area
         USING IFBNT_TOKEN,R2          Set addressability
*                                      DSNLOGREC TOKEN area
         L     R2,IFBNT_LOGREC_NAME_PTR  Get pointer to data set name
         DROP  R2                      Free up register 2
         USING IFBNT_LOGREC,R2         Set addressability to
*                                      LOGREC data set name area
*********************************************************************
* If you are interested in obtaining the log stream name, reference
* IFBNT_LOGREC_LOGSTREAM instead of IFBNT_LOGREC_NAME here,
* using the MVC command to move the log stream name to your
* own program's area.
*********************************************************************
         MVC   LOGRNAME,IFBNT_LOGREC_NAME  Move LOGREC data set name
*                                      to own area
         DROP  R2                      Free up register 2
EXIT     DS    0H                      Return point
         SLR   R15,R15                 Set return code of zero
         PR                            Return to caller
         EJECT
ABEND    ABEND X'BAD'                  ABEND if non-zero return code
         EJECT
*********************************************************************
* Local working storage declares
*********************************************************************
NAME     DS    CL16                    Name for Name/Token pair
TOKEN    DS    XL16                    Token for Name/Token Pair
RETCODE  DS    F                       Return code from IEANTRT
LOGRNAME DS    CL44                    Area for LOGREC data set name
*
```

```
*********************************************************************
* Constant and Equates
*********************************************************************
LEVEL    DC    A(IEANT_SYSTEM_LEVEL)   SYSTEM LEVEL
R0       EQU   0
R1       EQU   1
R2       EQU   2
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         EJECT
*********************************************************************
* NAME/TOKEN SYSTEM LEVEL DSNLOGREC VARIABLE DECLARES
*********************************************************************
         IFBNTASM
         EJECT
*********************************************************************
* NAME/TOKEN VARIABLE DECLARES
*********************************************************************
         IEANTASM
         END   IFBNTXMP
```

Chapter 48. IEANTRTR — Name/token retrieve register interface

Description

The IEANTRTR macro provides the name/token retrieve service in a way that does not require the user to have storage prior to the call.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller can hold a local, CML, or CMS lock; however, no locks are required.
Control parameters:	Input and output parameters can, for AMODE 64 callers, be above 2G. The NAME parameter is assumed to be within the primary address space. The LEVEL and TOKEN parameters can, for AR-mode callers, be in an ALET-qualified space.

Programming requirements

Before you use name/token services, you can optionally include macro IEANTASM which includes name/token service equates.

Restrictions

- Do not use the IEANTRTR service unless you know that you are running on a release with z/OS V2R2 functions (as indicated by bit CVTZOS_V2R2 in the CVT data area being on).
- Do not use the IEANTRTR callable service in a RESMGR resource manager routine unless one of the following is true:
  - The name/token pair is a system-level persistent name/token pair.
  - The resource manager is running for a daughter task of the task that owns the name/token pair.
  - The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

- Do not call the IEANTRTR callable service with *user_name* and *user_token* parameters being the same storage locations.

Input register information

Before issuing the IEANTRTR macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IEANTRTR macro, the caller does not have to place any information into any AR unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
When GR15=0, 64-bit GR0 contains bytes 0-7 of the token of the requested name/token pair. Otherwise, used as a work register by the system.

1
When GR15=0, 64-bit GR1 contains bytes 8-15 of the token of the requested name/token pair. Otherwise, used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IEANTRTR macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.

Syntax	Description
␣	One or more blanks must precede IEANTRTR.
IEANTRTR	
␣	One or more blanks must follow IEANTRTR.
LEVEL=*level*	*level*: RX-type address or address in register (2) - (12)
,NAME=*name*	*name:* RX-type address or address in register (2) - (12)
,TOKEN=*token*	*token* RX-type address or address in register (2) - (12)
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEANTRTR macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

LEVEL=*level*
A required input parameter that specifies the level of the name/token pair from which you want to retrieve the token (each of these levels has a corresponding equate in IEANTASM):

1
Task

Equate name: IEANT_TASK_LEVEL

2
Home address space

Equate name: IEANT_HOME_LEVEL

3
Primary address space

Equate name: IEANT_PRIMARY_LEVEL

4
System

Equate name: IEANT_SYSTEM_LEVEL

11
Task with authorization check

Equate name: IEANT_TASKAUTH_LEVEL

12
Home address space with authorization check

Equate name: IEANT_HOMEAUTH_LEVEL

13
Primary address space with authorization check.

Equate name: IEANT_PRIMARYAUTH_LEVEL

Note: Levels 11 (IEANT_TASKAUTH_LEVEL), 12 (IEANT_HOMEAUTH_LEVEL), and 13 (IEANT_PRIMARYAUTH_LEVEL) indicate that the IEANTRTR callable service should determine if the name/token pair being retrieved was created by an authorized program.

To code: Specify the RX-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,NAME=*name*
A required input parameter that specifies the name of the requested name/token pair.

To code: Specify the RX-type address, or address in register (2)-(12), of a 16-character field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,TOKEN=*token*
An optional output parameter that is to contain the token of the requested name/token pair. The token is also in 64-bit GRs 0 and 1.

To code: Specify the RX-type address, or address in register (2)-(12), of a 16-character field.

ABEND codes

None.

Return codes

When IEANTRTR returns control to your program, GPR 15 (and *return_code*, when you specify RETCODE) contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends as action that you need to take.

Table 55. Return Codes for the IEANTRTR Macro

Hexadecimal Return Code	Equate Symbol	Meaning	Action
00	IEANT_OK	The operation was successful.	None.
04	IEANT_NOT_FOUND	The request is rejected because the system could not find the requested name/token pair.	Check the token *name* you specified.
16	IEANT_NOT_AUTH	A request for a retrieval with authorization check attempted to retrieve a name/token pair created by an unauthorized caller.	If your program is authorized, you need to make sure that the name/token pair you are retrieving was created by another authorized program. You may choose to use the name/token pair if it was created by an unauthorized program, but doing so might cause data integrity problems.

Table 55. Return Codes for the IEANTRTR Macro (continued)			
Hexadecimal Return Code	**Equate Symbol**	**Meaning**	**Action**
20	IEANT_SRB_MODE	An SRB-mode caller attempted to retrieve a task-level name/token pair.	Check which level of name/token pair you are retrieving.
28	IEANT_LEVEL_INVALID	The caller specified an incorrect *level*.	Specify the correct *level*. Valid options are 1, 2, 3, 4, 11, 12, or 13.
64	IEANT_UNEXPECTED_ERROR	A system error occurred while handling the request.	Retry the request.

Example 1

Operation: Retrieve the token, but leave it in registers 0 and 1.

The code is as follows:

```
************************************************************
* Retrieve the task level token for MY_NAME                *
************************************************************
         IEANTRTR LEVEL=L,NAME=lName
         LTR   15,15           Check retcode
         JNZ   Error
*        Token in GR0,1 is valid
         ....
lName    DC   CL16'MY_NAME'
L        DC   A(IEANT_TASK_LEVEL)
         IEANTASM ,
```

Example 2

Operation: Save the token into a variable.

The code is as follows:

```
************************************************************
* Retrieve the system level token for MY_SYSNAME and       *
* save the token                                           *
************************************************************
         IEANTRTR LEVEL=L,NAME=lName,TOKEN=T
         LTR   15,15           Check retcode
         JNZ   Error
*        Token in T is valid
         ....
lName    DC   CL16'MY_SYSNAME'
T        DS   CL16
L        DC
A(IEANT_SYSTEM_LEVEL)
         IEANTASM ,
```

Chapter 49. IEAN4CR — Create a name/token pair

Description

Call the IEAN4CR service to create a name/token pair.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key. **Note:** Problem-state programs with PSW key 8 - 15 cannot create system-level pairs.
Dispatchable unit mode:	Task or SRB **Note:** SRB-mode callers cannot create a task-level pair.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	The parameter list and all parameters must reside in the caller's primary address space.

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

```
* Name/Token Level Constants
*
IEANT_TASK_LEVEL             EQU     1
IEANT_HOME_LEVEL             EQU     2
IEANT_PRIMARY_LEVEL          EQU     3
IEANT_SYSTEM_LEVEL           EQU     4
IEANT_TASKAUTH_LEVEL         EQU     11
IEANT_HOMEAUTH_LEVEL         EQU     12
IEANT_PRIMARYAUTH_LEVEL      EQU     13
*
* Name/Token Persistence Constants
*
IEANT_NOPERSIST              EQU     0
IEANT_PERSIST                EQU     1
IEANT_NOCHECKPOINT           EQU     0
IEANT_CHECKPOINTOK           EQU     2
*
* Name/Token Return Code Constants
*
IEANT_OK                     EQU     0
IEANT_DUP_NAME               EQU     4
IEANT_NOT_FOUND              EQU     4
IEANT_24BITMODE              EQU     8
IEANT_NOT_AUTH               EQU     16
IEANT_SRB_MODE               EQU     20
IEANT_LOCK_HELD              EQU     24
IEANT_LEVEL_INVALID          EQU     28
```

```
IEANT_NAME_INVALID          EQU     32
IEANT_PERSIST_INVALID       EQU     36
IEANT_AR_INVALID            EQU     40
IEANT_UNEXPECTED_ERR        EQU     64
```

Restrictions

Do not use the IEAN4CR callable service in a RESMGR resource manager routine unless one of the following is true:

- The name/token pair is a system-level persistent name/token pair.
- The resource manager is running for a daughter task of the task that owns the name/token pair.
- The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information

Before issuing the IEAN4CR callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEAN4CR	,(level ,user_name ,user_token ,persist_option ,return_code)

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEAN4CR:

```
1.   LOAD EP=IEAN4CR
     Save the 8-byte entry point address with bit 63 changed to 0
     (...)
     Put the saved entry point address with bit 63 changed to 0 into 64-bit R15
     CALL (15),(...)

2.   LLGT  15,X'10'
     L     15,X'220'(15,0)
     L     15,X'14'(15,0)
     L     15,X'7C'(15,0)
     CALL  (15),(...)
```

Both of these alternate techniques require verification that the IEAN4CR service is available (in the CVT, bit CVTZOS_V1R11 is on indicating that the program is running on z/OS V1R11 or a later release).

Parameters

The parameters are explained as follows:

level
Specifies a fullword that contains an integer indicating the level of the name/token pair:

1
Task

2
Home address space

3
Primary address space

4
System.

,user_name
Specifies the 16-byte area containing the name of the name/token pair that the user creates. The bytes of the name may have any value. The name may contain blanks, integers, or addresses.

Names must be unique within a level. Here are some examples.

- Two task-level name/token pairs owned by the same task cannot have the same name. However, two task-level name/token pairs owned by different tasks can have the same name.
- Two home-address-space-level name/token pairs in the same address space cannot have the same name. However, two home-address-space-level name/token pairs in different address spaces can have the same name.

Because of these unique requirements you must avoid using the same names that IBM uses for name/token pairs. Do not use the following names:

- Names that begin with A through I
- Names that begin with X'00'.

,user_token
Specifies the 16-byte area containing the token of the name/token pair that the user creates.

,persist_option
Specifies a fullword that contains an integer indicating if a system-level name/token pair should persist after the creating address space's job step task terminates or if Checkpoint/Restart can be issued if the program has this task-level name/token pair. If a program has non-task-level name/token pairs or has task-level name/token pairs that did not specify IEANT_CHECKPOINTOK, the program cannot take a checkpoint.

- 0 - system-level pair will not persist and checkpoint is not permitted.
- 1 - system-level pair will persist.
- 2 - checkpoint is permitted.

Note: Only system-level name/token pairs can persist after the creating task terminates. Only task-level name/token pairs can permit checkpoint.You must specify 0 for all other levels.

,return_code
Specifies a fullword to contain the return code from the IEAN4CR service.

ABEND codes

The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See *z/OS MVS System Codes* for an explanation and responses for these codes.

Return and reason codes

When IEAN4CR returns control to your program, GPR 15 and *return_code* contain a return code. The following table identifies return codes in hexadecimal and decimal, tells what each means, and recommends an action that you need to take:

Table 56. Return Codes for the IEAN4CR Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The operation was successful. **Action**: None.
04	**Meaning**: The *user_name* specified already exists. **Action**: Choose a different *user_name*.
08	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 64-bit addressing mode.
10	**Meaning**: An unauthorized caller attempted to create a system-level name/token pair. **Action**: Check which level of name/token pair you are creating.
14	**Meaning**: An SRB-mode caller attempted to create a task-level name/token pair. **Action**: Change your program to task mode or use a different level.
18	**Meaning**: The caller held locks. **Action**: Release all locks before issuing IEAN4CR.
1C	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid options are 1, 2, 3, or 4.
20	**Meaning**: The caller specified an incorrect *user_name*. **Action**: Respecify the correct *user_name*.

Table 56. Return Codes for the IEAN4CR Macro (continued)

Hexadecimal Return Code	Meaning and Action
24	**Meaning**: The caller specified an incorrect *persist_option*. **Action**: • For system-level name/token pairs, you must specify zero or one for the *persist_option*. • For task-level name/token pairs, you must specify zero or two for the *persist_option*. • For home or primary address space level name/token pairs, you must specify zero for the *persist_option*.
28	**Meaning**: The caller was in AR ASC mode and AR1 was not zero. **Action**: Change your program to primary mode or set AR1 to zero.
40	**Meaning**: A system error occurred while handling the request. **Action**: Retry the request.

Chapter 50. IEAN4DL — Delete a name/token pair

Description

Call the IEAN4DL service to delete a name/token pair.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key **Note:** Problem-state programs with PSW key 8 - 15 cannot delete: • System-level pairs • Name/token pairs created by supervisor-state or PSW key 0-7 programs.
Dispatchable unit mode:	Task or SRB **Note:** SRB-mode callers cannot delete a task-level pair.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	The parameter list and all parameters must reside in the caller's primary address space.

Programming requirements

Before you use name/token services, you can optionally include the IEANTASM macro to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

```
* Name/Token Level Constants
*
IEANT_TASK_LEVEL             EQU     1
IEANT_HOME_LEVEL             EQU     2
IEANT_PRIMARY_LEVEL          EQU     3
IEANT_SYSTEM_LEVEL           EQU     4
IEANT_TASKAUTH_LEVEL         EQU     11
IEANT_HOMEAUTH_LEVEL         EQU     12
IEANT_PRIMARYAUTH_LEVEL      EQU     13
*
* Name/Token Persistence Constants
*
IEANT_NOPERSIST              EQU     0
IEANT_PERSIST                EQU     1
*
* Name/Token Return Code Constants
*
IEANT_OK                     EQU     0
IEANT_DUP_NAME               EQU     4
IEANT_NOT_FOUND              EQU     4
IEANT_24BITMODE              EQU     8
IEANT_NOT_AUTH               EQU     16
IEANT_SRB_MODE               EQU     20
```

```
IEANT_LOCK_HELD              EQU     24
IEANT_LEVEL_INVALID          EQU     28
IEANT_NAME_INVALID           EQU     32
IEANT_PERSIST_INVALID        EQU     36
IEANT_AR_INVALID             EQU     40
IEANT_UNEXPECTED_ERR         EQU     64
```

Restrictions

Do not use the IEAN4DL callable service in a RESMGR resource manager routine unless one of the following is true:

- The name/token pair is a system-level persistent name/token pair.
- The resource manager is running for a daughter task of the task that owns the name/token pair.
- The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.

Input register information

Before issuing the IEAN4DL callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEAN4DL	,(level ,user_name ,return_code)

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEAN4DL:

```
1.   LOAD EP=IEAN4DL
     Save the 8-byte entry point address with bit 63 changed to 0
     (...)
     Put the saved entry point address with bit 63 changed to 0 into 64-bit R15
     CALL (15),(...)

2.   LLGT  15,X'10'
     L     15,X'220'(15,0)
     L     15,X'14'(15,0)
     L     15,X'84'(15,0)
     CALL  (15),(...)
```

Both of these alternate techniques require verification that the IEAN4DL service is available (in the CVT, bit CVTZOS_V1R11 is on indicating that the program is running on z/OS V1R11 or a later release).

Parameters

The parameters are explained as follows:

level
Specifies a fullword that contains an integer indicating the level of the name/token pair you wish to delete:

1
Task

2
Home address space

3
Primary address space

4
System.

,user_name
Specifies the 16-byte area containing the name of the name/token pair to be deleted.

,return_code
Specifies a fullword to contain the return code from the IEAN4DL service.

ABEND codes

The caller might encounter abend X'AC7' with a reason code of either X'00030000' or X'00030001'. See *z/OS MVS System Codes* for an explanation and responses for these codes.

Return and reason codes

When IEAN4DL returns control to your program, GPR 15 and *return_code* contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends an action that you need to take.

Table 57. Return Codes for the IEAN4DL Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The operation was successful. **Action**: None.
04	**Meaning**: The request is rejected because the system could not find the requested name/token pair. **Action**: Check the *user_name* you specified.
08	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 64-bit addressing mode.
10	**Meaning**: An unauthorized caller attempted to delete a system-level pair or a name/token pair that was created by an authorized program. **Action**: Check which level of name/token pair you are deleting.
14	**Meaning**: An SRB-mode caller attempted to delete a task-level name/token pair. **Action**: Change the program to task mode or check the value you set for the level parameter.
18	**Meaning**: The caller held locks. **Action**: Release all locks before issuing IEAN4DL.
1C	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid options are 1, 2, 3, or 4.
20	**Meaning**: The caller specified an incorrect *user_name*. **Action**: Respecify the correct *user_name*.
28	**Meaning**: The caller was in AR ASC mode and AR1 was not zero. **Action**: Change your program to primary mode or set AR1 to zero.
40	**Meaning**: A system error occurred while handling the request. **Action**: Retry the request.

Chapter 51. IEAN4RT — Retrieve the token from a name/token pair

Description

Call the IEAN4RT service to retrieve the token from a name/token pair.

The IEAN4RT callable service can also be used to obtain the name of the logrec medium, either the name of the logrec data set or the name of the logrec log stream.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key
Dispatchable unit mode:	Task or SRB **Note:** SRB-mode callers cannot retrieve a task-level pair.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller can hold a local, CML, or CMS lock; however, no locks are required.
Control parameters:	The parameter list and all parameters must reside in the caller's primary address space.

Programming requirements

Before you use name/token services, you can optionally include macro IEANTASM to invoke name/token services equate (EQU) statements. IEANTASM provides the following constants for use in your program:

```
* Name/Token Level Constants
*
IEANT_TASK_LEVEL             EQU     1
IEANT_HOME_LEVEL             EQU     2
IEANT_PRIMARY_LEVEL          EQU     3
IEANT_SYSTEM_LEVEL           EQU     4
IEANT_TASKAUTH_LEVEL         EQU     11
IEANT_HOMEAUTH_LEVEL         EQU     12
IEANT_PRIMARYAUTH_LEVEL      EQU     13
*
* Name/Token Persistence Constants
*
IEANT_NOPERSIST              EQU     0
IEANT_PERSIST                EQU     1
*
* Name/Token Return Code Constants
*
IEANT_OK                     EQU     0
IEANT_DUP_NAME               EQU     4
IEANT_NOT_FOUND              EQU     4
IEANT_24BITMODE              EQU     8
IEANT_NOT_AUTH               EQU     16
IEANT_SRB_MODE               EQU     20
IEANT_LOCK_HELD              EQU     24
```

```
IEANT_LEVEL_INVALID          EQU     28
IEANT_NAME_INVALID           EQU     32
IEANT_PERSIST_INVALID        EQU     36
IEANT_AR_INVALID             EQU     40
IEANT_UNEXPECTED_ERR         EQU     64
```

To obtain the name of the logrec data set or the name of the logrec log stream, you can include the IFBNTASM macro, as well as the IEANTASM macro, in your program. See "Example 2" on page 582 for the list of definitions IFBNTASM provides.

Restrictions

- Do not use the IEAN4RT callable service in a RESMGR resource manager routine unless one of the following is true:
  - The name/token pair is a system-level persistent name/token pair.
  - The resource manager is running for a daughter task of the task that owns the name/token pair.
  - The resource manager is running for the task that owns the name/token pair and that resource manager was established for a specific address space and a specific task.
- Do not call the IEAN4RT callable service with *user_name* and *user_token* parameters being the same storage locations.

Input register information

Before issuing the IEAN4RT callable service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEAN4RT	,(level ,user_name ,user_token ,return_code)

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEAN4RT:

```
1.   LOAD EP=IEAN4RT
     Save the 8-byte entry point address with bit 63 changed to 0
     (...)
     Put the saved entry point address with bit 63 changed to 0 into 64-bit R15
     CALL (15),(...)

2.   LLGT  15,X'10'
     L     15,X'220'(15,0)
     L     15,X'14'(15,0)
     L     15,X'80'(15,0)
     CALL  (15),(...)
```

Both of these alternate techniques require verification that the IEAN4RT service is available (in the CVT, bit CVTZOS_V1R11 is on indicating that the program is running on z/OS V1R11 or a later release).

Parameters

The parameters are explained as follows:

level
Specifies a fullword that contains an integer indicating the level of the name/token pair from which you want to retrieve the token:

1
Task

2
Home address space

3
Primary address space

4
System

11
Task with authorization check

12
Home address space with authorization check

13
Primary address space with authorization check.

Note: Levels 11, 12, and 13 indicate that the IEAN4RT callable service should determine if the name/token pair being retrieved was created by an authorized program.

,user_name
Specifies the 16-byte area containing the name of the requested name/token pair.

,user_token
Specifies the 16-byte area to contain the token of the requested name/token pair.

,return_code
Specifies a fullword to contain the return code from the IEAN4RT service.

ABEND codes

None.

Return codes

When IEAN4RT returns control to your program, GPR 15 and *return_code* contain a return code. The following table identifies return codes in hexadecimal, tells what each means, and recommends as action that you need to take.

Table 58. Return Codes for the IEAN4RT Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The operation was successful. **Action**: None.
04	**Meaning**: The request is rejected because the system could not find the requested name/token pair. **Action**: Check the *user_name* you specified.
08	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 64-bit addressing mode.
10	**Meaning**: A request for a retrieval with authorization check attempted to retrieve a name/token pair created by an unauthorized caller. **Action**: If your program is authorized, you need to make sure that the name/token pair you are retrieving was created by another authorized program. You may choose to use the name/token pair if it was created by an unauthorized program, but doing so might cause data integrity problems.
14	**Meaning**: An SRB-mode caller attempted to retrieve a task-level name/token pair. **Action**: Check which level of name/token pair you are retrieving.
1C	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid options are 1, 2, 3, 4, 11, 12, or 13.
40	**Meaning**: A system error occurred while handling the request. **Action**: Retry the request.

Chapter 52. IEARBUP — RB update service

Description

IEARBUP allows you to request that the system update the instruction address in the PSW copy in the RB.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks are required. The caller may hold a local lock, the CMS lock or the CPU lock.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

The caller must include the CVT and IHAECVT mapping macros.

Restrictions

If the caller holds the CPU lock, the parameter list must be in fixed or DREF storage.

Input register information

Before issuing the IEARBUP macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IEARBUP macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEARBUP.
	IEARBUP
ƀ	One or more blanks must follow IEARBUP.
WHICHRB=CURRENT	
WHICHRB=PREV	
WHICHRB=EXPLICIT	
,RB=*xrb*	
FUNCTION=UPDATE	**Default:** FUNCTION=UPDATE
,PSWBYTE03=NO	
,PSWBYTE03=YES	
,ADDRTYPE=NO_CHANGE	**Default:** ADDRTYPE=NO_CHANGE
,ADDRTYPE=INRBOPSWA	

Syntax	Description
,ADDRTYPE=ACTUAL	
,PSWADDR=*pswaddr*	*pswaddr:* RS-type address or address in register (2) - (12)
,AMODE=UNCHANGED	**Default:** AMODE=UNCHANGED
,AMODE=24	
,AMODE=31	
,AMODE=64	
,ADDRTYPE=DELTA	
,PSWDELTA=*pswdelta*	*pswdelta:* RS-type address or address in register (2) - (12)
FUNCTION=EXTRACTPSW	
,PSWG=*xpswg*	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEARBUP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

WHICHRB=
A required parameter that identifies the RB to be updated

WHICHRB=CURRENT
indicates to update the current RB.

WHICHRB=PREV
indicates to update the previous (older) RB.

WHICHRB=EXPLICIT
indicates to update the provided RB. The calling program must ensure that there is proper serialization to keep the provided RB valid for the duration of IEARBUP service processing.

RB=*xrb*
indicates the name (RS-type) or address in register (2)-(12) of a required character input that identifies the RB to be updated

FUNCTION=
indicates an optional keyword input that identifies the function to be performed

FUNCTION=UPDATE
indicates to update an RB

,PSWBYTE03=
A required parameter that indicates whether the user has updated the first 4 bytes (bytes 0 to 3) of RBOPSW. If so, the system should use those updated values.

,PSWBYTE03=NO
indicates that bytes 0 to 3 were **not** modified.

,PSWBYTE03=YES
indicates that bytes 0 to 3 were modified.

,ADDRTYPE=
An optional parameter that identifies the method by which the instruction address in the PSW is provided. The default is ADDRTYPE=NO_CHANGE.

,ADDRTYPE=NO_CHANGE
indicates that the instruction address has not been changed.

,ADDRTYPE=INRBOPSWA
indicates that the instruction address has been updated in RBOPSWA, along with the one or more AMODE indicators.

,ADDRTYPE=ACTUAL
indicates that the instruction address is to be used as is.

,ADDRTYPE=DELTA
indicates that the value provided is a delta to the existing address.

,PSWDELTA=*pswdelta*
When ADDRTYPE=DELTA is specified, a required input parameter that contains the delta to be added to the instruction address in the PSW copy stored in the RB. The value is treated as a signed quantity, so a value of X'FFFFFFFE' would be treated as negative two, resulting in subtracting two from the instruction address. The AMODE will remain unchanged.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,PSWADDR=*pswaddr*
When ADDRTYPE=ACTUAL is specified, a required input parameter that contains the address to be placed into the PSW stored in the RB. The high 33 bits must be zero, unless the result is to be AMODE 64.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-byte field.

,AMODE=
When ADDRTYPE=ACTUAL is specified, a required parameter that identifies the resulting AMODE for the PSW

,AMODE=UNCHANGED
indicates not to change the AMODE.

,AMODE=24
indicates to set the AMODE to 24.

,AMODE=31
indicates to set the AMODE to 31.

,AMODE=64
indicates to set the AMODE to 64.

FUNCTION=EXTRACTPSW
indicates to extract the 128–bit PSW associated with this RB

PSWG=*xpswg*
indicates the name (RS-type) or address in register (2)-(12) of the required 16–character output that is to contain the 128–bit x/Architecture PSW.

Note: If running under ESA/390 architecture (ARCHLVL 1), the PSW is the 128–bit z/Architecture analog of the 64–bit ESA/390 PSW.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

,PLISTERVER=IMPLIED_VERSION
indicates the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

,PLISTVER=MAX
indicates the parameter list will be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

,PLISTVER=0
indicates that you want to use the currently available parameters.

,MF=
An optional input parameter that specifies the macro form.

,MF=S
Specifies the **standard** form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

,MF=L,*list addr*
Specifies the **list** form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

,list addr
The name (RS-type) or address in register (1)-(12) of the storage area that contains the parameters.

,attr
An optional 1- to 60-byte input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,MF=E,*list addr*,COMPLETE
Specifies the **execute** form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name (RS-type) or address in register (1)-(12) of the storage area that contains the parameters.

COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

The caller may get the following abend code:

0C2-02
The caller was not in supervisor state.

0C4-04
The caller was not in key 0.

Return and reason codes

When the IEARBUP macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro IEARBUPM provides equate symbols for the return and reason codes.

Table 59 on page 610 identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 59. Return and Reason Codes for the IEARBUP Macro

Return Code	Reason Code	Equate Symbol, Meaning, and Action
0	—	**Equate Symbol**: IearbupRc_OK **Meaning**: Iearbup request successful.
8	—	**Equate Symbol**: IearbupRc_InvParm **Meaning**: Iearbup request specifies invalid parameters. **Action**: Refer to the action provided with the specific reason code.

Table 59. Return and Reason Codes for the IEARBUP Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol, Meaning, and Action**
8	xxxx0801	**Equate Symbol**: IearbupRsnBadVersion **Meaning**: The version field in the parameter list is not valid. **Action**: Check for possible storage overlay.
8	xxxx0802	**Equate Symbol**: IearbupRsnBadAMODEField **Meaning**: The amode field in the parameter list is not valid. **Action**: Check for possible storage overlay.
8	xxxx0803	**Equate Symbol**: IearbupRsnBadAddress **Meaning**: The address provided is not valid. **Action**: Only provide an instruction address that is less than X'80000000'.
C	—	**Equate Symbol**: IearbupRc_Env **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	xxxx0C01	**Equate Symbol**: IearbupRsnPrevRBNotFound **Meaning**: RB=PREV was requested, but there is only one RB for the current task. **Action**: Use RB=CURRENT when there is only one RB.
C	xxxx0C02	**Equate Symbol**: IearbupRsnBadAMODE **Meaning**: AMODE=64 was specified but the architecture level is not z/Architecture. **Action**: Only request AMODE=64 when the architecture level is z/Architecture.

Example 1

Operation

1. Update the instruction address in the PSW copy stored in the RB to the address provided in field P.

The code is as follows:

```
      IEARBUP ADDRTYPE=ACTUAL,PSWADDR=P,RETCODE=RC,MF=(E,MFL)
         ...
      IEARBUP MF=(L,MFL)
P     DS    XL8
RC    DS    F
```

Example 2

Operation:

1. Decrement the instruction address in the PSW copy in the RB by 4

The code is as follows:

```
      IEARBUP ADDRTYPE=DELTA,PSWDELTA=PD,RETCODE=RC,MF=(E,MFL)
         ...
      IEARBUP MF=(L,MFL)
PD    DC    F'-4'
RC    DS    F
```

Chapter 53. IEATDUMP — Transaction dump request

Description

Transaction dump is a service used to request an unformatted dump of virtual storage to a data set, similar to a SYSMDUMP. It is invoked with the IEATDUMP assembler macro, which issues SVC 51. The service is available to both authorized and unauthorized callers; however, not all functions are available to unauthorized callers. If an unauthorized caller requests a transaction dump with authorized keywords, the request will be rejected and message IEA820I will be issued indicating this condition. The transaction dump can be written to one or more automatically allocated data sets by specifying a data set name pattern, similar to the pattern used for the operator DUMPDS NAME=parameter. Automatic allocation reduces the exposure that a dump is truncated because of space constraints, and is done using the generic allocation unit name of SYSALLDA. When a dump is written, messages IEA822I or IEA827I are issued indicating whether the dump is complete or partial.

When a transaction dump is written, a dump directory record describing the dump may be written. The dump directory to be used is specified on the dump request using the IDX keyword. If no dump directory is specified on the request, the directory allocated to IPCSDDIR in the current job step will be used. If no dump directory is specified and IPCSDDIR is not allocated, no record describing the dump will be written.

Dump suppression occurs using symptoms available in the current SDWA or a symptom string may be provided (via the SYMREC keyword). If a symptom string is provided and an SDWA exists, the symptom string is used for suppression purposes. Statistics for dump suppression are contained in the DAE data set and are not differentiated from SYSMDUMPs. If a dump is requested but not taken because it was suppressed, message IEA820I is issued indicating this condition.

Authorized users may specify the REMOTE keyword, on a transaction dump invocation, to request that other address spaces on the current or other MVS images (in the same sysplex) be dumped. When remote dumps are requested, automatic allocation must also be used. Transaction dump uses an incident token to associate this dump with other diagnostic information. Automatic allocation also uses this incident token for symbol substitution in the data set name pattern. An incident token may be generated using the IEAINTKN macro and provided on the dump request using the INTOKEN keyword. If an incident token is not provided, one will be generated and used internally. While an incident token may always be specified, it may be especially important when remote dumps are requested.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15. Use of some keywords is restricted to authorized callers (supervisor state, PSW key 0-7 or APF-authorized).
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not hold any locks.

Environmental factor	Requirement
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The caller-provided title, data set name, dump index name, symptom record, incident token, remote area, problem description area and storage list area all have the same requirements and restrictions as the control parameters.

Programming requirements

None.

Restrictions

The caller may not have any FRRs established.

An IEATDUMP cannot succeed when another process within the task is exclusively holding the SYSZTIOT enqueue. Instead, a SVC dump would probably occur.

Input register information

Before issuing the IEATDUMP macro, the caller does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the IEATDUMP macro, the caller does not have to place any information into any access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-14
Unchanged

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The parameters DCB, DCBAD, and ASYNC=YES are no longer supported.

The IEATDUMP macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEATDUMP.
IEATDUMP	
␣	One or more blanks must follow IEATDUMP.
DSNAD=*dsnad*	*dsnad*: RS-type address or register (2) - (12).
DSN=*dsn*	*dsn*: RS-type address or register (2) - (12).
DDNAME=*ddname*	*ddname*: RS-type address or register (2) - (12).
DSNTEST=NO	Default: DSNTEST=NO
DSNTEST=YES	
,HDRAD=*hdrad*	*hdrad*: RS-type address or register (2) - (12).
,HDR=*hdr*	*hdr*: RS-type address or register (2) - (12).
,IDXAD=*idxad*	*idxad*: RS-type address or register (2) - (12).
,IDX=*idx*	*idx*: RS-type address or register (2) - (12).
,SYMRECAD=*symrecad*	*symrecad*: RS-type address or register (2) - (12).
,SYMREC=*symrec*	*symrec*: RS-type address or register (2) - (12).
,INTOKENAD=*intokenad*	*intokenad*: RS-type address or register (2) - (12).
,INTOKEN=*intoken*	*intoken*: RS-type address or register (2) - (12).
,REMOTEAD=*remotead*	*remotead*: RS-type address or register (2) - (12).
,REMOTE=*remote*	*remote*: RS-type address or register (2) - (12).

Syntax	Description
,PROBDESCAD=*probdescad*	*probdescad*: RS-type address or register (2) - (12).
,PROBDESC=*probdesc*	*probdesc*: RS-type address or register (2) - (12).
,LISTAD=*listad*	*listad*: RS-type address or register (2) - (12).
,LIST=*list*	*list*: RS-type address or register (2) - (12).
,SUBPLSTAD=*subplstad*	*subplstad*: RS-type address or register (2) - (12).
,SUBPLST=*subplst*	*subplst*: RS-type address or register (2) - (12).
,DSPLISTAD=*dsplistad*	*dsplistad*: RS-type address or register (2) - (12).
,DSPLIST=*dsplist*	*dsplist*: RS-type address or register (2) - (12).
,SDATA=DEFS	**Default**: SDATA=DEFS
,SDATA=ALLNUC	
,SDATA=CSA	
,SDATA=GRSQ	
,SDATA=LPA	
,SDATA=LSQA	
,SDATA=NUC	
,SDATA=RGN	
,SDATA=SQA	
,SDATA=SUM	
,SDATA=SWA	
,SDATA=TRT	
,SDATA=PSA	
,ASYNC=NO	**Default**: ASYNC=NO
,ECBAD=*ecbad*	*ecbad*: RS-type address or register (2) - (12).
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).

Syntax	Description
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	*list addr*: RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters DCB, DCBAD, and ASYNC=YES are no longer supported, and are removed from this information.

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEATDUMP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

DSNAD=*dsnad*
DSN=*dsn*
DDNAME=*ddname*
A required input parameter. The output dump data set should have the attributes of RECFM=FB and LRECL=4160.

DSNAD=*dsnad*
A 4-byte field which contains the address of the area of the name pattern used to create the data set that is to contain the dump. The format of the area is described in the DSN field which follows.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

DSN=*dsn*
A 2- to 101-character input area that contains the name pattern used to create the data set that is to contain the dump. The format of the area begins with a single byte specifying the length of the name pattern, which must not be greater than 100. The name pattern immediately follows that byte. The name pattern has a series of attributes: it is similar to that used by the operator DUMPDS NAME= parameter, except that &SEQ is not supported, and there is no default name pattern available; the use of system symbols is supported; and it must resolve to a valid data set name which can be allocated from the caller's task. When used with the REMOTE= parameter, the generated name must be unique for each requested address space (&JOBNAME is one recommended addition to the pattern to accomplish this).

In addition, IEATDUMP also recognizes the symbol &DS. (Dump Section) on the end of the name pattern. When present, IEATDUMP allocates the first data set for dumping, ending with “001”. If this runs out of disk space or uses up all 16 extents before the dump is completed, dumping will be continued to data sets with the same name, but ending in “002”,”003”, and so on, until the entire dump is written. Each of these data sets are allocated with a primary extent size of 500M and a secondary extent size of 500M, but it is possible to change these values by providing ACS routines that are driven by DFSMS.

Remember to combine all of the data sets into one data set by using IPCS COPYDUMP, before using IPCS to view the diagnostic data.

To code: Specify the RS-type address, or address in register (2) - (12), of a 2- to 101-character field.

DDNAME=*ddname*
An 8-character input field that is the name of the DD representing the data set that is to contain the dump. The DD must be allocated when IEATDUMP is invoked. The system will open this DD.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

DSNTEST={NO|YES}
An optional keyword input that determines whether a Transaction dump will be attempted, or that only the DSN or DSNAD specified name pattern will be tested. On systems without the support, executing programs run as if DSNTEST=NO was specified. The RtctDsnTest bit of the IHARTCT macro will indicate if the support is available.

DEFAULT=NO

DSNTEST=NO
The default specifying that a Transaction dump should be attempted. Processing is identical to that when the parameter is not specified.

DSNTEST=YES
Transaction dump processing should only attempt to allocate the data set using the DSN or DSNAD specified name pattern

,HDRAD=*hdrad*
,HDR=*hdr*
A required input parameter.

,HDRAD=*hdrad*
A 4-byte field which contains the address of a parameter of the dump title. The format of the area is a single byte specifying the length of the title followed by the title itself.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,HDR=*hdr*
A 2- to 101-character input area that contains the dump title. The format of the area is a single byte specifying the length of the title followed by the title itself. The title has a maximum length of 100 characters.

To code: Specify the RS-type address, or address in register (2) - (12), of a 2- to 101-character field.

,IDXAD=*idxad*
,IDX=*idx*
An optional input parameter.

,IDXAD=*idxad*
A 4-byte field which contains the address of a parameter of an area that contains the name of the dump index which is to contain information about the dump after the dump is written. The format of the area is a single byte specifying the length of the dump index data set name followed by the name itself. The data set must be an existing IPCS dump directory. The data set will be allocated from the caller's address space.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,IDX=*idx*
A 2- to 45-character input area that contains the name of the dump index which is to contain information about the dump after the dump is written. The format of the area is a single byte specifying the length of the dump index data set name followed by the name itself. The name of the dump index data set has a maximum length of 44 characters. The data set must be an existing IPCS dump directory. The data set will be allocated from the caller's address space.

To code: Specify the RS-type address, or address in register (2) - (12), of a 2- to 45-character field.

,SYMRECAD=*symrecad*
,SYMREC=*symrec*
An optional input parameter.

,SYMRECAD=*symrecad*
A 4-byte field which contains the address of a parameter of a valid symptom record for DAE to use for dump suppression. This area is built using SYMRBLD and mapped by ADSR. This area has a maximum length of 1900 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,SYMREC=*symrec*
A parameter of a valid symptom record for DAE to use for dump suppression. This area is built using SYMRBLD and mapped by ADSR. This area has a maximum length of 1900 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,INTOKENAD=*intokenad*
,INTOKEN=*intoken*
An optional input parameter.

,INTOKENAD=*intokenad*
A 4-byte field which contains the address of a parameter of a 32-byte area that contains an incident token previously built by the IEAINTKN macro.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,INTOKEN=*intoken*
A parameter of a 32-byte area that contains an incident token previously built by the IEAINTKN macro.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,REMOTEAD=*remotead*
,REMOTE=*remote*
An optional input parameter.

,REMOTEAD=*remotead*
A 4-byte field which contains the address of an area that identifies other address spaces to be dumped. This keyword is restricted to authorized callers. The format of the area is described in the REMOTE parameter which follows.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,REMOTE=*remote*
An optional character input area that can be a maximum of 1024 bytes long, which identifies other address spaces to be dumped. This keyword is restricted to authorized callers. The address spaces can be on the current system and/or other systems in the sysplex. The area is mapped by the IHASDRMT mapping macro. Through IHASDRMT, you can identify the systems to be dumped and specify the content of the dumps on individual systems. One can also specify that the following parameters on the IEATDUMP macro be copied for the remote dumps requested: SDATA, DSPLIST, and SUBPLST. The area consists of:

- A 4-byte header, which indicates the total length of the area. The length must include the four bytes of the header.

- Contents entry. Each entry consists of:

  ID
  A 2-byte field, whose value identifies the content type. The values are declared by the constants with names beginning with SDRMT_IDCON in the IHASDRMT mapping.

  Length
  A 2-byte field that gives the length of the contents portion. The length must include the 2 bytes of this length field, plus the 2 bytes of the ID field.

  Contents
  A variable field that gives the contents identified in the ID field. The contents you can specify are the system names, job names, XCF group and member names, data space names, address space identifiers, SDATA options, storage ranges, subpools, and keys. Within the contents, the following items also support the use of wildcards:

  – System name
  – Job name
  – XCF group name
  – XCF member name
  – Data space name and its qualifying job name

  See wildcard support under the description of the SDUMPX macro.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,PROBDESCAD=*probdescad*
,PROBDESC=*probdesc*
An optional input parameter.

,PROBDESCAD=*probdescad*
A 4-byte field which contains the address of a parameter of an area that contains information describing the problem. This area has a maximum length of 1024 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,PROBDESC=*probdesc*
A parameter of an area that contains information describing the problem. This area has a maximum length of 1024 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,LISTAD=*listad*
,LIST=*list*
An optional input parameter.

,LISTAD=*listad*
A 4-byte field which contains the address of a parameter of a list of starting and ending addresses of areas to be dumped. The high-order bit of the last ending address is set to 1; the high-order bit of all other addresses is 0. This area has a maximum length of 240 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,LIST=*list*
A parameter of a list of starting and ending addresses of areas to be dumped. The high-order bit of the last ending address is set to 1; the high-order bit of all other addresses is 0. This area has a maximum length of 240 bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,SUBPLSTAD=*subplstad*
,SUBPLST=*subplst*
An optional input parameter.

,SUBPLSTAD=*subplstad*
A 4-byte field which contains the address of a parameter of a list of subpool numbers to be dumped. The first halfword is the number subpools in the list and must be on a fullword boundary. Each entry is two bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,SUBPLST=*subplst*
A parameter of a list of subpool numbers to be dumped. The first halfword is the number subpools in the list and must be on a fullword boundary. Each entry is two bytes.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,DSPLISTAD=*dsplistad*
,DSPLIST=*dsplist*
An optional input parameter.

,DSPLISTAD=*dsplistad*
A 4-byte field which contains the address of a parameter of a list of data space storage to be dumped. The first word is the total size of the DSPLIST. The next eight characters is the STOKEN of the data space to be dumped. A full word indicates the number of ranges to be dumped for that STOKEN. Then, 2 full words for each range, which are the starting and ending addresses of the range. More than one STOKEN may be specified per DSPLIST.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,DSPLIST=*dsplist*
A parameter of a list of data space storage to be dumped. The first word is the total size of the DSPLIST. The next eight characters is the STOKEN of the data space to be dumped. A full word indicates the number of ranges to be dumped for that STOKEN. Then, 2 full words for each range, which are the staring and ending addresses of the range. More than one STOKEN may be specified per DSPLIST.

To code: Specify the RS-type address, or address in register (2) - (12), of a character field.

,SDATA=DEFS
,SDATA=ALLNUC
,SDATA=CSA
,SDATA=GRSQ
,SDATA=LPA
,SDATA=LSQA
,SDATA=NUC
,SDATA=RGN
,SDATA=SQA
,SDATA=SUM
,SDATA=SWA
,SDATA=TRT
,SDATA=PSA
An optional parameter that specifies what system data should be provided in the transaction dump. No fetch-protected storage which is inaccessible in the caller's key will be dumped. The default is SDATA=DEFS.

,SDATA=DEFS
The following SDATA options are included in the dump: LSQA, NUC, PSA, RGN, SQA, SUM, SWA, and TRT.

,SDATA=ALLNUC
All of DAT-on nucleus, including page-protected areas, and all of the DAT-off nucleus.

,SDATA=CSA
Common storage area and virtual storage for 64-bit addressable memory objects created using one of the following services:

- `IARV64 REQUEST=GETCOMMON,DUMP=LIKECSA`

- IARCP64 COMMON=YES,DUMP=LIKECSA
- IARST64 COMMON=YES,TYPE=PAGEABLE

,SDATA=GRSQ
Global resource serialization (ENQ/DEQ/RESERVE) queues.

,SDATA=LPA
Link pack area for this job.

,SDATA=LSQA
Local system queue area and virtual storage for 64-bit addressable memory objects created using one of the following services:

- IARV64 REQUEST=GETSTOR,DUMP=LIKELSQA
- IARCP64 COMMON=NO,DUMP=LIKELSQA
- IARST64 COMMON=NO

,SDATA=NUC
Non-page-protected areas of the DAT-on nucleus.

,SDATA=RGN
Entire private area including virtual storage for 64-bit addressable memory objects created using the following services:

- IARV64 REQUEST=GETSTOR,DUMP=LIKERGN
- IARV64 REQUEST=GETSTOR,SVCDUMPRGN=YES
- IARCP64 COMMON=NO,DUMP=LIKERGN
- IARST64 COMMON=NO

,SDATA=SQA
System queue area and virtual storage for 64-bit addressable memory objects created using one of the following services:

- IARV64 REQUEST=GETCOMMON,DUMP=LIKESQA
- IARCP64 COMMON=YES,DUMP=LIKESQA
- IARST64 COMMON=YES,TYPE=FIXED
- IARST64 COMMON=YES,TYPE=DREF

,SDATA=SUM
Requests the summary dump function.

,SDATA=SWA
Scheduler work area.

,SDATA=TRT
System trace data.

,SDATA=PSA
Prefixed save area.

One or more values may be specified for the SDATA parameter. If more than one value is specified, group the values within parentheses.

,ASYNC=NO
An optional parameter that specifies whether the transaction dump should be taken synchronously. The default is ASYNC=NO.

,ASYNC=NO
The transaction dump should be taken synchronously.

,ECBAD=*ecbad*
,ECB=*ecb*
An optional input parameter.

,ECBAD=*ecbad*
A 4-byte field which contains the address of a parameter of an ECB to be posted when the entire dump has been written. This area must be on a word boundary.

To code: Specify the RS-type address, or address in register (2) - (12), of a pointer field.

,ECB=*ecb*
A parameter of an ECB to be posted when the entire dump has been written. This area must be on a word boundary.

To code: Specify the RS-type address, or address in register (2) - (12), of a 4-character field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the

execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of IEATDUMP in the following order:

- Use IEATDUMP ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use IEATDUMP ...MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
- Use IEATDUMP ...MF=(E,list-addr,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IEATDUMP macro returns control to your program:

- GPR 15 (and *retcode*, if you coded RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, if you coded RSNCODE) contains a reason code.

X'00000000'
A complete dump was written.

X'00000004'
A partial dump was written.

X'00000008'
No dump was written.

X'0000000C'
Internal processing error. No dump was written.

X'00000010'
Unexpected return code from IEAVAD00.

Table 60. Return and Reason Codes for the IEATDUMP Macro

Return Code	Reason Code	Meaning and Action
00000000	00000000	**Meaning**: A complete dump was written. **Action**: None.
00000004	00000001	**Meaning**: The dump was truncated because the data set was too small. **Action**: Reissue IEATDUMP with a larger data set or use the DSN|DSNAD parameter to allocate the dump data set automatically.
00000004	00000002	**Meaning**: Contention detected when attempting to set tasks in the address space non-dispatchable. **Action**: Data in dump may be inconsistent. Reissue IEATDUMP.
00000004	00000003	**Meaning**: Unable to add dump data set to dump index. **Action**: Verify that the dump index specified on the IDX parameter is correct and reissue IEATDUMP.
00000004	00000004	**Meaning**: Unable to allocate transaction dump data set. **Action**: See allocation failure messages. Reissue IEATDUMP.
00000004	00000006	**Meaning**: Maximum amount of dump sections reached (999). **Action**: Dump less memory, or use ACS routines to increase the size of the data sets. Reissue IEATDUMP.
00000004	00000007	**Meaning**: The system has filled one of the range tables. **Action**: Dump less memory. If the problem still exists, contact the IBM Support Center.
00000004	00000008	**Meaning**: The data space used for the IEATDUMP has been filled. No more then 2 gigabytes of data can be collected. **Action**: Do one of the following: • Remove unnecessary dump options. • Specify smaller memory ranges. • Use the extended data set support by either: – Preallocating a large capacity data set and use the DDNAME parameter. – Refer to the use of the &DS symbol for the DSN parameter's data set name pattern.
00000008	00000000	**Meaning**: For DSNTEST=YES, a dump data set was successfully allocated and deleted. Hence, no dump data set is available, but the pattern is useful. Typically, no dump data set is available, but the pattern should be useful. **Action**: The dump data set pattern resulted in the successful allocation of a data set. Normal IEATDUMP requests should not fail simply because the specified name pattern conflicts with installation conventions.
00000008	00000001	**Meaning**: The address of the transaction dump parameter list was zero. **Action**: Ensure register 1 is non-zero when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000002	**Meaning**: The dump was suppressed by CHNGDUMP. **Action**: Issue CHNGDUMP SET,SYSMDUMP or CHNGDUMP RESET,SYSMDUMP. Reissue IEATDUMP.
00000008	00000003	**Meaning**: The dump was suppressed by SLIP. **Action**: Delete SLIP trap with SLIP DEL command. Reissue IEATDUMP.
00000008	00000004	**Meaning**: The ALET for the transaction dump parameter list was not valid. **Action**: Ensure that access register 1 has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.

Table 60. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
00000008	00000005	**Meaning**: The transaction dump parameter list was not addressable. **Action**: Ensure that the entire transaction dump parameter list is addressable via register 1 (and access register 1 if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000006	**Meaning**: The transaction dump parameter list version number was not valid. **Action**: Ensure the transaction dump request was built using the IEATDUMP macro for the system on which the dump was requested. Reissue IEATDUMP.
00000008	00000007	**Meaning**: The length of the transaction dump parameter list did not match the parameter list version number. **Action**: Ensure the transaction dump request was built using the IEATDUMP macro for the system on which the dump was requested. Reissue IEATDUMP.
00000008	00000008	**Meaning**: No DDNAME, DSN(AD), or DSP_STOKEN was specified. **Action**: Reissue IEATDUMP with the DDNAME, DSN(AD) or DSP_STOKEN keyword.
00000008	00000009	**Meaning**: Both DDNAME and DSN(AD) keywords were specified. **Action**: Reissue IEATDUMP with either the DDNAME or DSN(AD) keyword.
00000008	0000000C	**Meaning**: The ALET for the DSN(AD) keyword was not valid. **Action**: Ensure that the access register for the DSN(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000000D	**Meaning**: The DSN(AD) was not addressable. **Action**: Ensure that the entire DSN(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000000E	**Meaning**: No HDR(AD) keyword was specified. **Action**: Reissue IEATDUMP with the HDR(AD) keyword.
00000008	0000000F	**Meaning**: The ALET for the HDR(AD) keyword was not valid. **Action**: Ensure that the access register for the HDR(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000010	**Meaning**: The HDR(AD) was not addressable. **Action**: Ensure that the entire HDR(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000011	**Meaning**: The specified HDR(AD) was longer than 100 characters. **Action**: Reissue IEATDUMP with a shorter header.
00000008	00000012	**Meaning**: The ALET for the IDX(AD) keyword was not valid. **Action**: Ensure that the access register for the IDX(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000013	**Meaning**: The IDX(AD) was not addressable. **Action**: Ensure that the entire IDX(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000014	**Meaning**: The IDX(AD) keyword did not specify a valid data set name after symbol substitution. **Action**: Reissue IEATDUMP with an IDX keyword that resolves to a valid dump index data set name.

Table 60. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
00000008	00000015	**Meaning**: The ALET for the SYMREC(AD) keyword was not valid. **Action**: Ensure that the access register for the SYMREC(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000016	**Meaning**: The SYMREC(AD) was not addressable. **Action**: Ensure that the entire SYMREC(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000017	**Meaning**: The specified SYMREC(AD) was not valid. Either ADSRID not set to 'SR' or primary symptom string offset or length not initialized. **Action**: Reissue IEATDUMP with a valid symptom record.
00000008	00000018	**Meaning**: The ALET for the INTOKEN(AD) keyword was not valid. **Action**: Ensure that the access register for the INTOKEN(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000019	**Meaning**: The INTOKEN(AD) was not addressable. **Action**: Ensure that the entire INTOKEN(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000001A	**Meaning**: The ALET for the REMOTE(AD) keyword was not valid. **Action**: Ensure that the access register for the REMOTE(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000001B	**Meaning**: The REMOTE(AD) was not addressable. **Action**: Ensure that the entire REMOTE(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000001C	**Meaning**: The specified REMOTE(AD) was not valid. **Action**: Reissue IEATDUMP with a valid remote area.
00000008	0000001D	**Meaning**: The ALET for the LIST(AD) keyword was not valid. **Action**: Ensure that the access register for the LIST(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000001E	**Meaning**: The LIST(AD) was not addressable. **Action**: Ensure that the entire LIST(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000001F	**Meaning**: The specified LIST(AD) was not valid. A range in the storage list had a start address greater than its ending address. **Action**: Reissue IEATDUMP with a valid storage list.
00000008	00000020	**Meaning**: The dump was rejected because the caller's authorization was insufficient for requested function(s). **Action**: Verify authorization and requested functions. Reissue IEATDUMP.
00000008	00000021	**Meaning**: The DSN(AD) keyword did not specify a valid data set name after symbol substitution. **Action**: Reissue IEATDUMP with a DSN keyword that resolves to a valid dump data set name.
00000008	00000022	**Meaning**: The DSN(AD) keyword specified a data set name that was too long. **Action**: Reissue IEATDUMP with a DSN(AD) keyword that resolves to a shorter dump data set name.

Table 60. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
00000008	00000023	**Meaning**: The DSN(AD) keyword specified a data set name that contained a bad symbol. **Action**: Reissue IEATDUMP with a DSN(AD) keyword that does not contain bad symbols.
00000008	00000024	**Meaning**: Unable to create data space to capture transaction dump. **Action**: Remedy cause of DSPSERV CREATE failure or request transaction dump specifying DDNAME or including the &DS. symbol in the DSN template.
00000008	00000025	**Meaning**: Unable to add transaction dump data space to access list. **Action**: Remedy cause of ALESERV ADD failure or request transaction dump specifying DDNAME. Reissue IEATDUMP.
00000008	00000026	**Meaning**: Unable to allocate transaction dump data set. **Action**: Look at allocation failure messages. Reissue IEATDUMP.
00000008	00000027	**Meaning**: The transaction dump was suppressed by DAE. **Action**: If you do not wish transaction dumps to be suppressed on an installation basis, issue the SET DAE=xx console command specifying an ADYSETxx member that does not specify SYSMDUMP(SUPPRESS). If you do not wish transaction dumps to be suppressed on an application basis, include the VRANODAE key in the VRADATA of your recovery routine. Reissue IEATDUMP.
00000008	00000028	**Meaning**: An error occurred writing the first record to the data space or dump data set. **Action**: Ensure the STOKEN and origin for the specified data space are correctly specified. Ensure that the specified DD is allocated when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000029	**Meaning**: The ALET for the PROBDESC(AD) keyword was not valid. **Action**: Ensure that the access register for the PROBDESC(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000002A	**Meaning**: The PROBDESC(AD) was not addressable. **Action**: Ensure that the entire PROBDESC(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000002B	**Meaning**: The specified PROBDESC(AD) was not valid. **Action**: Reissue IEATDUMP with a valid problem description area.
00000008	0000002C	**Meaning**: The ALET for the SUBPLST(AD) keyword was not valid. **Action**: Ensure that the access register for the SUBPLST(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000002D	**Meaning**: The SUBPLST(AD) was not addressable. **Action**: Ensure that the entire SUBPLST(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000002E	**Meaning**: The specified SUBPLST(AD) was not valid. An invalid subpool was specified. **Action**: Reissue IEATDUMP with a valid subpool list.
00000008	0000002F	**Meaning**: The ALET for the DSPLIST(AD) keyword was not valid. **Action**: Ensure that the access register for the DSPLIST(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.

Table 60. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
00000008	00000030	**Meaning**: The DSPLIST(AD) was not addressable. **Action**: Ensure that the entire DSPLIST(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000031	**Meaning**: The specified DSPLIST(AD) was not valid. An invalid data space was specified. **Action**: Reissue IEATDUMP with a valid data space list.
00000008	00000032	**Meaning**: The ALET for the ECB(AD) keyword was not valid. **Action**: Ensure that the access register for the ECB(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000033	**Meaning**: The ECB(AD) was not addressable. **Action**: Ensure that the entire ECB(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.
00000008	00000034	**Meaning**: The specified ECB(AD) was not valid. The ECB was not on a fullword boundary. **Action**: Reissue IEATDUMP with an ECB.
00000008	00000035	**Meaning**: OPEN failed for the dump data set. **Action**: Determine why OPEN failed and reissue IEATDUMP.
00000008	00000036	**Meaning**: Dump data set has invalid block size. **Action**: Correct the block size and reissue IEATDUMP.
00000008	00000037	**Meaning**: The DSP_RECORDS@ field was not accessible. **Action**: Correct the problem and reissue IEATDUMP.
00000008	00000038	**Meaning**: The DCB parameter is not supported on IEATDUMP. **Action**: Remove the DCB parameter and reissue IEATDUMP.
00000008	00000039	**Meaning**: The ASYNC=YES is not supported on IEATDUMP. **Action**: Change to ASYNC=NO and reissue IEATDUMP.
00000008	0000003A	**Meaning**: The &DS. symbol was found in the midst of the dump DSN name pattern. **Action**: Place the &DS symbol at the end of the DSN name pattern and reissue IEATDUMP.
00000008	0000003B	**Meaning**: This IEATDUMP was not taken because another dump was already running in the address space. **Action**: None.
00000008	0000003C	**Meaning**: For DSNTEST=YES, a dump data set could not be allocated. **Action**: The dump data set name pattern specified via the DSN or DSNAD parameter resulted in a data set name that conflicts with installation conventions. Change the pattern to adhere to those conventions.
0000000C	00000001	**Meaning**: Unable to obtain storage for transaction dump from subpool 230 below the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	00000002	**Meaning**: Unable to establish recovery environment for transaction dump. **Action**: Determine why ESTAEX failed and reissue IEATDUMP.

Table 60. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
0000000C	00000003	**Meaning**: Unable to obtain storage for transaction dump from subpool 245 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	00000004	**Meaning**: Unable to obtain storage for transaction dump from subpool 231 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	00000005	**Meaning**: Unable to obtain storage for transaction dump from subpool 239 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	00000006	**Meaning**: Unable to obtain storage for transaction dump from subpool 239 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	00000007	**Meaning**: Unable to obtain storage for transaction dump from subpool 239 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	00000008	**Meaning**: Unable to obtain storage for transaction dump from subpool 250 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	00000009	**Meaning**: Unable to obtain storage for transaction dump from subpool 230 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	0000000A	**Meaning**: Unable to obtain storage for transaction dump from subpool 230 below the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	0000000B	**Meaning**: Unable to obtain storage for transaction dump from subpool 253 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	0000000C	**Meaning**: Unable to obtain high virtual private storage for transaction dump. **Action**: Determine why an authorized IARV64 MEMLIMIT(NO) request failed to get storage and reissue IEATDUMP.
0000000C	0000000D	**Meaning**: Unable to obtain high virtual common storage for transaction dump. **Action**: Determine why storage is not available and reissue IEATDUMP.
0000000C	000000FF	**Meaning**: IEAVTDMP's recovery received control. One possible reason is that the SYSZTIOT enqueue is being held exclusively by another process running under this task. It is not possible for the IEATDUMP to successfully complete. **Action**: The assistance of a system programmer is needed for associated SVC dumps. In the case of a SYSZTIOT enqueue, the problem is not in IEATDUMP processing. The diagnosis of any issues requires data collection using SLIP and/or SDUMPX, and not IEATDUMP.
00000010	xxxxxxxx	**Meaning**: Unexpected return code from IEAVAD00. Return code from IEAVAD00 returned as reason code. **Action**: Inform the system programmer.

Examples

An example using DSN:

```
         IEATDUMP DSN=DUMPDSN,HDR=DUMPTTL2
         .
         .
         .
DUMPDSN  DC    AL1(E2-S2)
S2       DC    C'HLQ.TDUMP.D&&YYMMDD..T&&HHMMSS..&&SYSNAME..&&JOBNAME.'
E2       EQU   *
DUMPTTL2 DC    AL1(E3-S3)
S3       DC    C'IEADUMP TO AUTOMATICALLY ALLOCATED DATA SET'
E3       EQU   *
```

Chapter 54. IEATEDS - Timed event data services

Description

IEATEDS provides timed event data services.

IEATEDS allows the user to record events to a Timed Event Data Table to provide information that will help determine flow and performance. Each event is time stamped and includes data provided by the caller and additional data collected by the service. A REXX exec is also provided to obtain a formatted report of the events.

To use the timed event data service:

1. Invoke IEATEDS with REQUEST=REGISTER to obtain and initialize a Timed Event Data Table. The size of the Timed Event Data Table is determined by the MaxEvents argument. Note that the Timed Event Data Table will not wrap. The REGISTER service will provide a Timed Event Data Token as output which will identify the newly created Timed Event Data Table on subsequent IEATEDS requests.
2. Invoke IEATEDS with REQUEST=RECORD, passing the Timed Event Data Token and other arguments, including up to 16 bytes of user data. Several RECORD requests may be made throughout the code to understand the flow and performance. Once the maximum number of events has been recorded, subsequent requests will be ignored.
3. Execute REXX exec IEAVFTED to output the Timed Event Data Report to a data set. The Timed Event Data Report parameters are described below and the format of the output is described in the IEATEDS macro example section. Note that IEAVFTED does not clear the Timed Event Data Table. Thus, IEAVFTED may be executed at any time to produce an up-to-date report containing all of the events that have been recorded so far.

Timed Event Data Report

The IBM supplied IEAVFTED REXX exec is used to produce a Timed Event Data Report in either a TSO or IPCS environment. When run under TSO, the user must specify either a pre-allocated data set or a z/OS UNIX file in which to place the report. The dataset option requires the user to allocate a data set with an LRECL of 512 and a RECFM of V or VB. The z/OS UNIX file option requires the TSO environment to have an OMVS segment. When run under IPCS, the Timed Event Data Report will normally be displayed within IPCS. An example is provided in the IEATEDS example section that shows how to have IPCS place the Timed Event Data Report into a pre-allocated data set.

The Timed Event Data Report will consist of two sections, the first section consisting of human readable text, and the second section consisting of spreadsheet data (unless the NOSS parameter, described below, is specified). Note that the IPL Statistics Table (IPST) will also be placed into the Timed Event Data Report.

The IEAVFTED REXX exec is a compiled REXX program which requires the full REXX compiler run-time libraries (at least REXX LIBR BASE MVS FMID HWJ9140) installed before attempting to use IEAVFTED. Note that IEAVFTED will not work with the REXX Alternate Runtime Library z/OS Base HWJ9143.

The IEAVFTED code resides in data set SYS1.SBLSCLI0. IEAVFTED must be run from a data set with an LRECL of 80 and a RECFM of F or FB.

The following describes the required and optional parameters for IEAVFTED:

- DATASET('*output_data_set*') or DA('*output_data_set*') is used to specify the name of the pre-allocated data set where the Timed Event Data Report will be written when IEAVFTED is run under TSO. The name must be fully qualified and the data set must have an LRECL of 512 with a RECFM of V or VB. Note that one and only one of DATASET('*output_data_set*'), DA('*output_data_set*'), or PATH('*z/OS UNIX file*') must be specified when IEAVFTED is run under TSO. Neither DATASET('*output_data_set*') nor DA('*output_data_set*') is allowed when IEAVFTED is run under IPCS (an example is provided in the

IEATEDS macro example section that shows how to have IPCS place the Timed Event Data Report into a pre-allocated data set).

- PATH('*z/OS UNIX file*') is used to specify the name of a z/OS UNIX file where the Timed Event Data Report will be written when IEAVFTED is run under TSO. A z/OS UNIX file is created along with its directories with the authority options of 770. Note that one and only one of PATH('*z/OS UNIX file*'), DATASET('*output_data_set*'), or DA('*output_data_set*') must be specified when IEAVFTED is run under TSO. PATH('*z/OS UNIX file*') is not allowed when IEAVFTED is run under IPCS as IPCS does not directly support putting output to a z/OS UNIX file.
- IPCSDA('*input_data_set*') is an optional specification that is used only with a special IEAVFTED invocation that will convert the spreadsheet data into a proper format in order to import it into a spreadsheet program. This is needed for the case where the Timed Event Data Report was obtained under IPCS using the example technique described in the IEATEDS macro example section. In that example technique, the output data set needs to be pre-allocated with an LRECL of 255 to satisfy IPCS, but in order to import the spreadsheet data into a spreadsheet program, the spreadsheet data needs to be in a 512 character data set or in a z/OS UNIX file. Invoking IEAVFTED with IPCSDA('*input_data_set*') does not produce a new report, but instead extracts the spreadsheet data contained in IPCSDA('*input_data_set*') and places it into the required data set as specified by one and only one of DATASET('*output_data_set*'), DA('*output_data_set*'), or PATH('*z/OS UNIX file*'). Parameters SS, NOSS, Component(*component_name*), and Comp(*component_name*) are ignored when IPCSDA is specified. Note that '*input_data_set*' and '*output_data_set*' must be different data set names.
- Component(*component_name*) or Comp(*component_name*) is optional and is used as a filter to obtain Timed Event Data Table information for a specific component. Each Timed Event Data Table is created with the component name specified as CompName on the IEATEDS REGISTER request. When component(*component_name*) is specified, the Timed Event Data Report will include Timed Event Data Table information only for those Timed Event Data Tables with a matching component name. Note that if the *component_name* value contains blanks, it must be enclosed within quotes. The *component_name* value is not case sensitive and will be converted to uppercase. Also, Component(IPST) can be used to obtain only the IPL Statistics Table. When neither Component() nor Comp() is specified, all Timed Event Data Tables for all components are included in the report. Note that Component() and Comp() are ignored when IPCSDA is specified.
- SS(*char*) is optional and is used to specify the character delimiter for the spreadsheet fields. The default character is a semicolon (;). Note that SS(*char*) is ignored when NOSS or IPCSDA are specified.
- NOSS is optional and is used to cause the spreadsheet data to be omitted from the Timed Event Data Report. Note that NOSS is ignored when IPCSDA is specified.
- HELP or ? is optional and is used to get a description of IEAVFTED and its parameters.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state. Any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any primary, any home, and any secondary address space
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller may hold a local lock, or a local lock and the CMS lock.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

The caller must include the IHAPSA, CVT, IHAECVT, and IHATEDS macros. Note that the IHATEDS macro has equate symbols for the return and reason codes, and for the length of the WorkArea.

Restrictions

None

Input register information

Before issuing the IEATEDS macro, the caller must ensure that general register 13 contains the address of a 216 byte save area. The save area must be in primary storage in the first 2G of storage. The caller does not have to place any information into any other general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

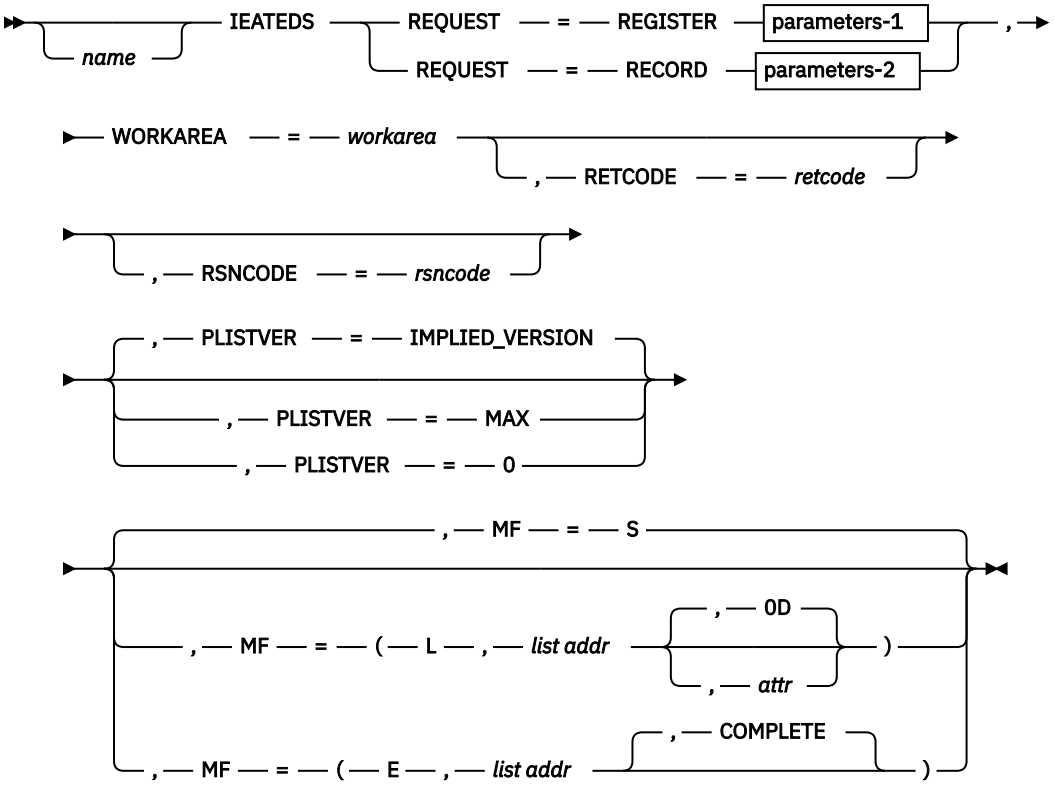
14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

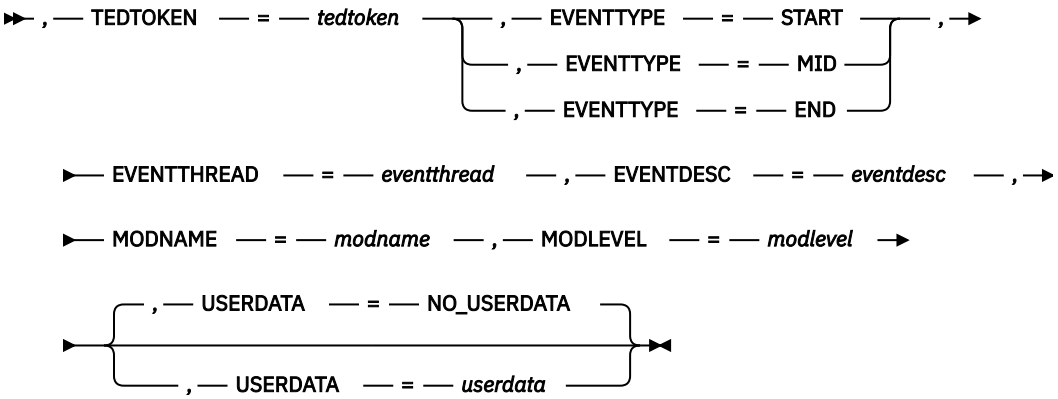
None

Syntax



parameters-1

parameters-2



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEATEDS macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,COMPNAME=*compname*
When REQUEST=REGISTER is specified, a required input parameter that specifies the component name that is registering. A mixed case value is supported.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,EVENTDESC=*eventdesc*
When REQUEST=RECORD is specified, a required input parameter that is used to describe the event. A mixed case value is supported.

To code: Specify the RS-type address, or address in register (2) - (12), of a 32-character field.

,EVENTTHREAD=*eventthread*
When REQUEST=RECORD is specified, a required input parameter that is used to provide an association for a series of events consisting of a start event, zero or more mid events, and an end event. This can be any character or hex value that the component finds useful. Using a unique value for each series of associated start, mid, and end events will help in understanding the flow and timing of the events. The Timed Event Data Report will include both the hex and the EBCDIC values for the thread. See the macro example section for more information regarding the Timed Event Data Report.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,EVENTTYPE=START
,EVENTTYPE=MID
,EVENTTYPE=END
When REQUEST=RECORD is specified, a required parameter that indicates the type of event to record.

,EVENTTYPE=START
The event is the start of a series of events.

,EVENTTYPE=MID
The event is one of a series of events. This mid event is matched to the start event whose EventThread matches the EventThread provided on this request.

,EVENTTYPE=END
The event is the last of a series of events. This end event is matched to the start event whose EventThread matches the EventThread provided on this request.

,MAXEVENTS=*maxevents*
When REQUEST=REGISTER is specified, a required input parameter that specifies the maximum number of events that will be recorded. This value will be used to determine the amount of storage to be allocated for the Timed Event Data Table for recording the events. The Timed Event Data Table size will be capped at 2M bytes, meaning that the value specified for MaxEvents will be reduced as necessary to a value where the Timed Event Data Table will be created within 2M bytes of storage. A return code and reason code will be returned if the MaxEvents was reduced, unless some other more serious error is returned.

Note also that all Timed Event Data Table storage is capped at 2G bytes, and any attempt to REGISTER once the 2G limit is reached will be rejected with a return code and reason code. No attempt will be made to reduce the MaxEvents in order to build a Timed Event Data Table any smaller than 2M just to get it to fit into the remaining storage that is near the 2G limit. Note also that there may be additional system controls or environmental conditions that limit this size to something smaller than 2G bytes.

The number of events that will fit into a 2M Timed Event Data Table is a function of the size of each entry. Since the entry size could grow over time, the number of events that will fit could be reduced in the future. This makes it difficult to accurately state the maximum number of events a Timed Event Data Table will hold, but a maximum of at least 2000 events is guaranteed.

Note that any attempt to record events beyond the resultant MaxEvents will not be recorded, but will be counted as an overflow count to assist in determining whether the number of RECORD requests should be decreased or MaxEvents should be increased (if not already at or above the maximum for a 2M Timed Event Data Table). The overflow count, requested MaxEvents, resultant MaxEvents, and the size of the resultant Timed Event Data Table can be found in the Timed Event Data Report which is described in the macro example section.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MODLEVEL=*modlevel*
When REQUEST=RECORD is specified, a required input parameter that specifies the module level that is recording this event.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,MODNAME=*modname*
When REQUEST=RECORD is specified, a required input parameter that specifies the module name that is recording this event.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long

enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

REQUEST=REGISTER
REQUEST=RECORD
A required parameter that indicates which service to perform.

REQUEST=REGISTER
Register the user for the timed event data service.

REQUEST=RECORD
Record the timed event data.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,TEDTOKEN=*tedtoken*
When REQUEST=REGISTER is specified, a required output parameter, whose returned value must be provided as input on subsequent REQUEST=RECORD calls.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,TEDTOKEN=*tedtoken*
When REQUEST=RECORD is specified, a required input parameter that is used to identify the timed event data collection to which this event shall be placed. This must be the TedToken that was returned as output from the Register request.

To code: Specify the RS-type address, or address in register (2) - (12), of a 16-character field.

,USERDATA=*userdata*
,USERDATA=NO_USERDATA
When REQUEST=RECORD is specified, an optional input parameter consisting of a comma delimited list of one or more variable names with a combined length (determined using L'varname for each variable) that does not exceed a total of 16 bytes. UserData may be any data that the user finds helpful in understanding the timed events. Note that the values must be simple items - for example, using substringed references is not allowed. The default is NO_USERDATA.

One or more values may be specified for the USERDATA parameter. If more than one value is specified, group the values within parentheses.

To code: Specify the RS-type address of a character field.

,WORKAREA=*workarea*
A required input parameter that specifies a work area on a double word boundary to be used by the timed event data service. The work area must be of size IEATEDS_WorkAreaSize (in macro IHATEDS) and must reside within the first 2 GB of primary storage in any key.

To code: Specify the RS-type address of a character field.

ABEND codes

0C2

Meaning: Privileged-operation exception. A problem state caller attempted to use IEATEDS.

Action: Get into supervisor state before invoking the function, perhaps by the MODESET macro.

Return and reason codes

When the IEATEDS macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro IHATEDS provides equate symbols for the return and reason codes. Note that the return and reason codes described below are hexadecimal values.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

Table 61. Return and reason codes for the IEATEDS macro

Return code	Reason code	Equate symbol meaning and action
0	—	**Equate symbol**: IEATEDSRc_OK **Meaning**: IEATEDS request was successful. **Action**: None required. **REGISTER** **Meaning**: The Timed Event Data Table was obtained and initialized and is ready for events to be recorded. **Action**: None required. **RECORD** **Meaning**: The event was placed into the Timed Event Data Table. **Action**: None required.
4	—	**Equate symbol**: IEATEDSRc_Warn **Meaning**: Warning **Action**: Refer to the action provided with the specific reason code.

Table 61. Return and reason codes for the IEATEDS macro (continued)

Return code	Reason code	Equate symbol meaning and action
4	00000401	**Equate symbol**: IEATEDSRsn_TedTableFull **Meaning**: An IEATEDS RECORD request was unable to place the new entry into the caller's Timed Event Data Table because the Timed Event Data Table is full. The Timed Event Data Table overflow count has been incremented. **Action**: Try increasing the value specified for MaxEvents on the IEATEDS REGISTER request. Note, however, that there is a limit of 2M bytes of storage for each Timed Event Data Table, and an overall limit of 2G bytes of storage for all of the Timed Event Data Tables in the system. You can execute the Timed Event Data Report REXX exec to examine the overflow count to determine the number of additional entries required to allow all of the RECORD requests to succeed. The Timed Event Data Report will also show the requested MaxEvents, the resultant MaxEvents, and the size of the Timed Event Data Table. If the size of the Timed Event Data Table is already at the 2M byte limit, try reducing the number of IEATEDS RECORD requests.
4	00000402	**Equate symbol**: IEATEDSRsn_MaxEventsReduced **Meaning**: An IEATEDS REGISTER request reduced the specified MaxEvents to allow the Timed Event Data Table to be built within the 2M-byte storage limit. **Action**: Execute the Timed Event Data Report REXX exec and examine the overflow count to determine whether the reduced MaxEvents has resulted in some RECORD requests not being recorded. If so, consider reducing the number of RECORD requests to a value that is no greater than the reduced MaxEvents (shown in the report as the resultant MaxEvents).
8	—	**Equate symbol**: IEATEDSRc_InvParm **Meaning**: IEATEDS request specified parameters that are not valid. **Action**: Refer to the action provided with the specific reason code.
8	00000801	**Equate symbol**: IEATEDSRsn_BadTedToken **Meaning**: An IEATEDS RECORD request supplied a TedToken value that was unable to locate a valid Timed Event Data Table. The RECORD request was not completed. **Action**: Ensure that the TedToken returned from the REGISTER request is not corrupted and is provided on the subsequent IEATEDS RECORD requests. Note that this error will also occur in the case of a Timed Event Data Table being corrupted or the storage becoming inaccessible. Execute the Timed Event Data Report REXX exec (described in the macro example section) to see whether the Timed Event Data Table in question was able to be located. If so, then the problem is with the specified TedToken.

Table 61. Return and reason codes for the IEATEDS macro (continued)

Return code	Reason code	Equate symbol meaning and action
C	—	**Equate symbol**: IEATEDSRc_Env **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	00000C01	**Equate symbol**: IEATEDSRsn_NoTedTableStorage **Meaning**: An IEATEDS REGISTER request was unable to obtain storage for the Timed Event Data Table. **Action**: Ensure that the system has enough above the bar common storage to satisfy a request for a Timed Event Data Table of the size being requested. Note that the maximum size allowed for a Timed Event Data Table is 2M bytes, and the maximum size for all Timed Event Data Table storage to 2G bytes. Note also that there may be additional system controls or environmental conditions that limit this size to something smaller than 2G bytes.
C	00000C02	**Equate symbol**: IEATEDSRsn_NoTedVectorTableStorage **Meaning**: An IEATEDS REGISTER request was unable to obtain storage for a Timed Event Data Vector Table. **Action**: Ensure that the system has enough above the bar common storage to satisfy a request for a Timed Event Data Vector Table which has a size of 4k.
10	—	**Equate symbol**: IEATEDSRc_CompError **Meaning**: Unexpected failure. **Action**: Contact your system programmer.
10	00001001	**Equate symbol**: IEATEDSRsn_UnexpectedError **Meaning**: An IEATEDS REGISTER or RECORD request had an unexpected error. The REGISTER of RECORD request completion status is unknown. **Action**: The system programmer should gather any diagnostic information that was produced and contact IBM support.

Examples

The following is an example of invoking IEATEDS to:

- Invoke IEATEDS to REGISTER
- Invoke IEATEDS to RECORD a Start event
- Invoke IEATEDS to RECORD a Mid event
- Invoke IEATEDS to RECORD an End event

The code is as follows.

```
         TITLE 'Sample code to register/record Timed Event Data'
TEDSAMPL CSECT
TEDSAMPL AMODE  31
TEDSAMPL RMODE  ANY
*/* START OF SPECIFICATIONS ********************************
```

```
*
*01* MODULE-NAME = TEDSAMPL
*
*02*   DESCRIPTIVE-NAME = Sample program to register and
*                         record Timed Event Data.
*01* DISCLAIMER =
*      This sample source 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.
*
**** END OF SPECIFICATIONS *******************************/
         EJECT
         BAKR  R14,0           Save on stack,return using r14
         BASR  12,0
         USING START,R12
START    EQU   *
         MODID ,
*************************************************************
*        START OF CODE
*************************************************************
         STORAGE OBTAIN,LENGTH=DynAreaLen,Addr=(R1),COND=NO, *
               LOC=ANY,SP=240
         LR R13,R1
         USING DynArea,R13
*************************************************************
*        Register for Timed Event Data Recording
*************************************************************
         IEATEDS Request=REGISTER,                          *
               CompName==CL32'TheProduct',                  *
               MaxEvents==F'64',                            *
               TedToken=TedToken,                           *
               WorkArea=TedWorkArea,                        *
               RetCode=RetCode,                             *
               RsnCode=RsnCode,                             *
               MF=(E,MyTedPLD,COMPLETE)
*
* Place code to check return/reason codes here.
*
*************************************************************
*        Record Timed Event Data for Event Thread SAMPLE
*************************************************************
         LA    R2,1
         ST    R2,DATA1
         MVC   DATA2(4),=CL4' RCD'
         IEATEDS Request=RECORD,                            *
               EventType=START,                             *
               EventThread==CL8'SAMPLE',                    *
               EventDesc==CL32'Timed Event Data sample',    *
               UserData=(Data1,Data2),                      *
               ModName==CL8'TEDSAMPL',                      *
               ModLevel==CL8'Level101',                     *
               TedToken=TedToken,                           *
               WorkArea=TedWorkArea,                        *
               RetCode=RetCode,                             *
               RsnCode=RsnCode,                             *
               MF=(E,MyTedPLD,COMPLETE)
*
* Place code to check return/reason codes here.
*
*************************************************************
*        Record Mid Timed Event Data
*************************************************************
         LA    R2,2
         ST    R2,DATA1
         MVC   DATA2(4),=CL4'XYZ1'
         MVC   DATA3(6),=CL6'FUNC 1'
         IEATEDS Request=RECORD,                            *
               EventType=MID,                               *
               EventThread==CL8'SAMPLE',                    *
               EventDesc==CL32'Before doing XYZ',           *
               UserData=(Data1,Data2,Data3),                *
               ModName==CL8'TEDSAMPL',                      *
               ModLevel==CL8'Level101',                     *
               TedToken=TedToken,                           *
               WorkArea=TedWorkArea,                        *
               RetCode=RetCode,                             *
               RsnCode=RsnCode,                             *
               MF=(E,MyTedPLD,COMPLETE)
*
```

```
* Place code to check return/reason codes here.
*
************************************************************
*        Record Last Timed Event Data for this Thread
************************************************************
         LA    R2,3
         ST    R2,DATA1
         MVC   DATA2(4),=CL4'XYZ1'
         MVC   DATA3(6),=CL6'FUNC 2'
         IEATEDS Request=RECORD,                             *
               EventType=END,                                *
               EventThread==CL8'SAMPLE',                     *
               EventDesc==CL32'After doing XYZ',             *
               UserData=(Data1,Data2,Data3),                 *
               ModName==CL8'TEDSAMPL',                       *
               ModLevel==CL8'Level101',                      *
               TedToken=TedToken,                            *
               WorkArea=TedWorkArea,                         *
               RetCode=RetCode,                              *
               RsnCode=RsnCode,                              *
               MF=(E,MyTedPLD,COMPLETE)
*
* Place code to check return/reason codes here.
*
************************************************************
*        Free Dynamic Area and Return
************************************************************
         LA    R0,DynAreaLen       Length of Dynamic Area
         STORAGE RELEASE,LENGTH=(R0),ADDR=(R13),SP=240

         PR
************************************************************
*        Dynamic Area
************************************************************
DynArea   DSECT
SaveArea  DS    XL216
TedToken  DS    XL16           Timed Event Data Token
Data1     DS    F              One word
Data2     DS    CL4            Four bytes
Data3     DS    CL6            Six Bytes
Data4     DS    CL2            Two Bytes
RetCode   DS    F              Return Code from Timed Event  x
                               Data Service
RsnCode   DS    F              Reason Code from Timed Event  x
                               Data Service
          DS    0D             Align TedWorkArea on dbl word
TedWorkArea DS  XL(IEATEDS_WORKAREASIZE)
           IEATEDS MF=(L,MyTedPLD)
DynAreaLen EQU *-DynArea       Length of DynArea
         IHATEDS               Constants and ret/rsn codes
************************************************************
*        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
         EJECT
************************************************************
* Mappings
************************************************************
         CVT DSECT=YES
         IHAPSA DSECT=YES
         IHAECVT
         END
```

Timed Event Data Report example invocations

Example invocations for TSO:

The following example invocation will produce a Timed Event Data Report with all components with spreadsheet data with the default spreadsheet delimiter of a semicolon. IBM may request that this data be sent to IBM for analysis. Note that *output_data_set* must be the name of a pre-allocated data set with an LRECL of 512 and a RECFM of V or VB.

```
IEAVFTED DA('output_data_set')
```

The following example invocation will produce a Timed Event Data Report for component ABC with spreadsheet data and with a spreadsheet delimiter of a question mark. Note that the component name is not case sensitive.

```
IEAVFTED DA('output_data_set(member)') COMPONENT(ABC) SS(?)
```

The following invocation will produce a Timed Event Data Report with all components without the spreadsheet data:

```
IEAVFTED DA('output_data_set') NOSS
```

The following invocation will produce a Timed Event Data Report to a z/OS UNIX file for component ABC with spreadsheet data: Note that the file name is case sensitive, the directories need not exist and COMP is abbreviated for COMPONENT.

```
IEAVFTED PATH('/usr/ted_data/Performance_Data_For_System_XYZ')
         COMP(ABC)
```

The following example JCL will run the IEAVFTED REXX exec in the TSO background. Note that the SYSEXEC data set must have an LRECL of 80 and a RECFM of F or FB.

```
//IEAVFTED JOB '123456,?',
// 'name',REGION=0M,
// MSGLEVEL=(1,1),CLASS=J,NOTIFY=name,
// MSGCLASS=H
//IEAVFTED EXEC PGM=IKJEFT01,ROLL=(NO,NO),DYNAMNBR=400,REGION=0M
//*************************************************************
//* Notes:
//*  - The REXX compiler run-time libraries must be installed.
//*
//*  - The data set containing the IEAVFTED exec
//*    (SYS1.SBLSCLI0) must have an LRECL of 80 and a RECFM of
//*    F or FB.
//*************************************************************
//SYSEXEC DD DISP=SHR,DSN=SYS1.SBLSCLI0
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN DD *
IEAVFTED DA('output_data_set')
/*
```

The following example JCL is a started procedure that will run the IEAVFTED REXX exec in the TSO background. Required and optional parameters for IEAVFTED are passed via the PARM keyword. See “Timed Event Data Report” on page 633 for the descriptions for the required and optional IEAVFTED parameters.

```
//TEDRPT     JOB MSGCLASS=A
//TEDRPT     PROC PARM='PATH(/usr/ted/ted_report)'
//  EXEC  PGM=IKJEFT01,
//   PARM='IEAVFTED &PARM'
//SYSEXEC  DD DSN=SYS1.SBLSCLI0,DISP=SHR
//SYSPROC  DD DSN=SYS1.PROCLIB,DISP=SHR
//SYSTSIN  DD DUMMY
//SYSTSPRT DD SYSOUT=*
//SYSINT   DD SYSOUT=(A,INTRDR),DCB=(LRECL=80,RECFM=FB)
//       PEND
//       EXEC TEDRPT
```

An example of a started procedure invocation using system symbolics as parameters:

```
S TEDRPT,PARM='PATH(/usr/&SYSPLEX/&SYSNAME/ted_from_12_1_2009)
  COMP(ABC)'
```

An example of a started procedure invocation specifying a TSO dataset and having NOSS (no spreadsheet data).

```
S TEDRPT,PARM='da(console.mttr.output(sample)) noss'
```

Example IPCS invocations:

The following example will produce a Timed Event Data Report with all components with spreadsheet data with a spreadsheet delimiter of a semicolon (the default). The output will be put into a data set that must be pre-allocated with an LRECL of 255 and a RECFM of V or VB. Note that IPCS only supports a maximum of 255 characters for the data set, but if the spreadsheet data needs to be imported into a spreadsheet program, then an additional procedure (described below) must be performed to extract the spreadsheet data from the 255 character data set and place it into either a pre-allocated 512 character data set or a z/OS UNIX file. Note that no data will be truncated or lost with the 255 character data set. The ALTLIB statement tells IPCS where to find the IEAVFTED exec. Note that the data set containing IEAVFTED (SYS1.SBLSCLI0) must have an LRECL of 80 and a RECFM of F or FB.

```
Create an output dsn (LRECL of 255, RECFM of V or VB)
TSO ALLOC F(IPCSPRNT) DA(dsn) SHR REUS
IP ALTLIB ACTIVATE APPL(EXEC) DA('SYS1.SBLSCLI0')
IP SETDEF PRINT NOTERM
IP IEAVFTED
IP CLOSE PRINT
IP SETDEF TERM NOPRINT
Data set dsn now contains the Timed Event Data Report
```

If the spreadsheet data needs to be imported into a spreadsheet program then one of the following IEAVFTED invocations must be issued from TSO. In the first example, the output_data_set must be pre-allocated with an LRECL of 512 and a RECFM of V or VB. In the second example, the output will go to a z/OS UNIX file. In both examples, the input_data_set is the 255 character data set from the above procedure for IPCS. IEAVFTED will extract the spreadsheet data from input_data_set, convert it into the proper format, and write it to the 512 character output_data_set or to the z/OS UNIX file, either of which can then be downloaded or FTP'd and imported into a spreadsheet program.

```
IEAVFTED IPCSDA('input_data_set') DA('output_data_set')
```

...or...

```
IEAVFTED IPCSDA('input_data_set') PATH('z/OS UNIX file')
```

Formatted Timed Event Data Report

The following example Timed Event Data Report was obtained by running the example TEDSAMPL assembler program described above, and then invoking IEAVFTED as follows:

```
IEAVFTED PATH(/usr/ted/example1) COMPONENT('THEPRODUCT')
```

```
*********************************************************************************************************
**************

*
            *
 * IBM z/OS Timed Event Data
Report                                                                                  *
 * Level: HBB7770-V1.03      Report Date/Time: 15 Mar 2010 16:01:23     Component Filter:
THEPRODUCT                 *
 * Sysplex: PLEX1       System: SY1       FMID: HBB7770  z/OS
V01R12M00                                  *
 * Machine: 4381-FF639F30 Online Standard CPs: 6 zAAPs: 0 zIIPs:
```

```
0                                                    *
 * IPL Start Date/Time: 15 Mar 2010
15:02:06.364187                                                              *

*
             *

*********************************************************************************************************
**************


*********************************************************************************************************
**************

*
             *
 * Total Timed Event Data Table Storage:
00072A70                                                          *

*
             *

*********************************************************************************************************
**************


*********************************************************************************************************
**************

*
             *
 * Timed Event Data Table - Component: TheProduct                        Address:
000001EF80605000                    *
 * Table Size: 00002C60      Register Date/Time: 15 Mar 2010
15:51:39.783511                                  *
 * Requested MaxEvents:         64  Resultant MaxEvents:    64  NumEvents: Current:     3
Overflow:          0        *

*
             *

*********************************************************************************************************
**************


 EntryNum:     1  Event Type/Thread: Start/E2C1D4D7D3C54040/*SAMPLE  *  Event Date/Time: 15 Mar 2010
15:51:39.783516
  Description: A sample of a TED entry
  HASN: 0025  PASN: 0025  Jobname: MAINASID  TCB: 005FF050  Module/Level/Offset: TEDSAMPL/
Level101/000000D0
  SRB/Task Time: 00000000002CFE00/000000000130A25C  User Data: 00000001 40D9C3C4 00000000 00000000 *....
RCD........*
  OUXBFCON: 00:00:00.334592  OUXBFDIS: 00:00:00.000000  OUXBFMNO: 00:00:00.052000  OUXBFWAIT:
00:00:00.134016
  Deltas:     IPL Start: 0 Days 00:49:33.419329    T.E.D. Registration: 0 Days 00:00:00.000005
     Thread Start Event: 0 Days 00:00:00.000000     Thread Prior Event: 0 Days 00:00:00.000000

 EntryNum:     2  Event Type/Thread: Mid  /E2C1D4D7D3C54040/*SAMPLE  *  Event Date/Time: 15 Mar 2010
15:51:39.783521
  Description: Before doing XYZ
  HASN: 0025  PASN: 0025  Jobname: MAINASID  TCB: 005FF050  Module/Level/Offset: TEDSAMPL/
Level101/00000148
  SRB/Task Time: 00000000002CFE00/000000000130A25C  User Data: 00000002 E7E8E9F1 C6E4D5C3 40F10000
*....XYZ1FUNC 1..*
  OUXBFCON: 00:00:00.334592  OUXBFDIS: 00:00:00.000000  OUXBFMNO: 00:00:00.052000  OUXBFWAIT:
00:00:00.134016
  Deltas:     IPL Start: 0 Days 00:49:33.419334    T.E.D. Registration: 0 Days 00:00:00.000010
     Thread Start Event: 0 Days 00:00:00.000004     Thread Prior Event: 0 Days 00:00:00.000004

 EntryNum:     3  Event Type/Thread: End  /E2C1D4D7D3C54040/*SAMPLE  *  Event Date/Time: 15 Mar 2010
15:51:39.783525
  Description: After doing XYZ
  HASN: 0025  PASN: 0025  Jobname: MAINASID  TCB: 005FF050  Module/Level/Offset: TEDSAMPL/
Level101/000001C6
  SRB/Task Time: 00000000002CFE00/000000000130A25C  User Data: 00000003 E7E8E9F1 C6E4D5C3 40F20000
*....XYZ1FUNC 2..*
  OUXBFCON: 00:00:00.334592  OUXBFDIS: 00:00:00.000000  OUXBFMNO: 00:00:00.052000  OUXBFWAIT:
00:00:00.134016
  Deltas:     IPL Start: 0 Days 00:49:33.419338    T.E.D. Registration: 0 Days 00:00:00.000014
     Thread Start Event: 0 Days 00:00:00.000009     Thread Prior Event: 0 Days 00:00:00.000004
```

```
*********************************************************************************************************
**************

*
            *
 * End Timed Event Data Table - Component:
TheProduct                                                                    *
 * Number Events: Start:    1  Mid:    1  End:
1                                                                  *

*
            *

*********************************************************************************************************
**************



*********************************************************************************************************
**************

*
            *
 * Description of
columns                                                                                              *
 *   Unique Id   : If multiple reports are merged into a single spread sheet, this field can be used to
uniquely       *
 *                 identify the source for the row of
data.                                                          *
 *   Event Time  : The time the event
occurred.                                                                        *
 *   Date        : The date the event
occurred.                                                                        *
 *   Event Thread: Used to tie the related Start/Mid/End events together. This can be a hex or EBCDIC
value.           *
 *   Thread EBCDIC: If Event Thread is in hex, this data is translated to EBCDIC. Unprintable characters
are           *
 *                 displayed as periods. The EBCDIC value is preceded and followed by an
asterisk.                  *
 *                 If the Event Thread is in EBCDIC, it is repeated in this
column.                                 *
 *   Type        : The type of event. (e.g. Start, Mid, End, Register,
Create).                               *
 *   Description : Text which describes the
event.                                                                  *
 *   Component   : The component to which this event
belongs.                                                     *
 *
Deltas
         *
 *     IPL Start : The amount of time that elapsed from the beginning of the IPL until the time of this
event.          *
 *     Thread Start Event: The amount of time that elapsed from the start event of the same Timed Event
Data table     *
 *                 and same Event Thread until the time of this
event.                                          *
 *     T.E.D. Registration: The amount of time that elapsed from the time the component registered
(created a Timed    *
 *                 Event Data table) until the time of this event. For the IPST data, this field will be
blank.         *
 *     Thread Prior Event: The amount of time that elapsed from the previous event of the same Timed
Event Data/IPS    *
 *                 table and same Event Thread until the time of this
event.                                        *
 *   Jobname     : The jobname of the address space that recorded this
event.                                        *
 *   HASN        : The home address space ID that recorded this
event.                                          *
 *   PASN        : The primary address space ID that recorded this
event.                                       *
 *   Module      : The name of the module that recorded this
event.                                         *
 *   Level       : The level of the module that recorded this event. For the IPST data, this field will
be blank.      *
 *   Offset      : The offset within the module where the event was recorded. For the IPST data, this
field will be    *
 *
blank.                                                                                                  *
 *   TCB@        : The TCB address that the module was running under when the event was recorded. For
the IPST data,   *
```

```
 *                this field will be blank. If the recording was done under an SRB, this field will be
zero.          *
 *   User1       : Data that the recording module wanted to record along with the
event.                                 *
 *   User2       : Data that the recording module wanted to record along with the
event.                                 *
 *   User3       : Data that the recording module wanted to record along with the
event.                                 *
 *   User4       : Data that the recording module wanted to record along with the
event.                                 *
 *   User EBCDIC : The four user data fields are displayed in EBCDIC. Unprintable characters are
displayed as          *
 *                periods. The data is preceded and followed by an asterisk. For the IPST data, this
field will be    *
 *
blank.                                                                                              *
 *   SRB         : The SRB time for the home address space where this event was recorded. For the IPST
data, this      *
 *                field will be
blank.                                                                                  *
 *   Task Time   : The task time for the home address space where this event was recorded. For the IPST
data, this     *
 *                field will be
blank.                                                                                  *
 *   OUXBFCON    : The time from the OUXBFCON field which contains the accumulated I/O FICON connect
time for the      *
 *                address space.  For the IPST data, this field will be
blank.                                          *
 *   OUXBFDIS    : The time from the OUXBFDIS field which contains the accumulated I/O FICON disconnect
time for the   *
 *                address space. For the IPST data, this field will be
blank.                                         *
 *   OUXBFMNO    : The time from the OUXBFMNO field which contains the FICON magic number - for every
I/O interrupt    *
 *                from a device attached to a FICON native CHPID, IOS will add one millisecond to this
field.         *
 *                For the IPST data, this field will be
blank.                                                      *
 *   OUXBFWAIT   : The time from the OUXBFWAIT field which contains the accumulated I/O FICON wait time
for the        *
 *                address space. This value includes pending time and control unit queue time. For the
IPST data,     *
 *                this field will be
blank.                                                                          *

*
             *
 * You may notice that some of the delta values (IPL Start, Thread Start ...) may be off by 0.000001
seconds. This is  *
 * due to rounding
effects.                                                                                            *

*
             *
 * By default, a semicolon is used as a column separator so when importing the data into a spreadsheet
program, the    *
 * data is placed in the correct columns. If any of the data contains a semicolon (or whatever the
separator character *
 * is), that semicolon will be replaced with a
blank.                                                        *

*
             *
 * When you import this data, remember to tell the program what the column separator character is so the
data is       *
 * formatted
correctly.
 *

*
             *

*********************************************************************************************************
**************



*********************************************************************************************************
**************

*
             *
```

```
 * IBM z/OS Timed Event Data
Report                                                                                *
 * Level: HBB7770-V1.03      Report Date/Time: 15 Mar 2010 16:01:23     Component Filter:
THEPRODUCT                  *

*
            *
 * The following is the same Timed Event Data/IPST as above but in a form that can be imported into a
spreadsheet      *
 * program. To do
so:                                                                                      *
 *   - Edit this data set (or a copy) and remove everything above the spreadsheet data (including these
directions.)    *
 *   - When you download or FTP the data set, specify ASCII or
TEXT.                                                  *
 *   - Do NOT download as
Binary.                                                                               *
 *   - When importing into the spreadsheet program, indicate that the data is delimited with a delimiter
character     *
 *
of: ;
       *

*
            *
 * Sorting by the 'Event Time' column will put all the data into chronological
order.                                  *

*
            *
 * Sorting by the 'Event Thread' and the 'Event Time' columns will group all the related events in
chronological       *
 * order. Examining the 'Thread Prior Event Delta' will help identify events that took a long
time.                    *

*
            *

*********************************************************************************************************
**************
```

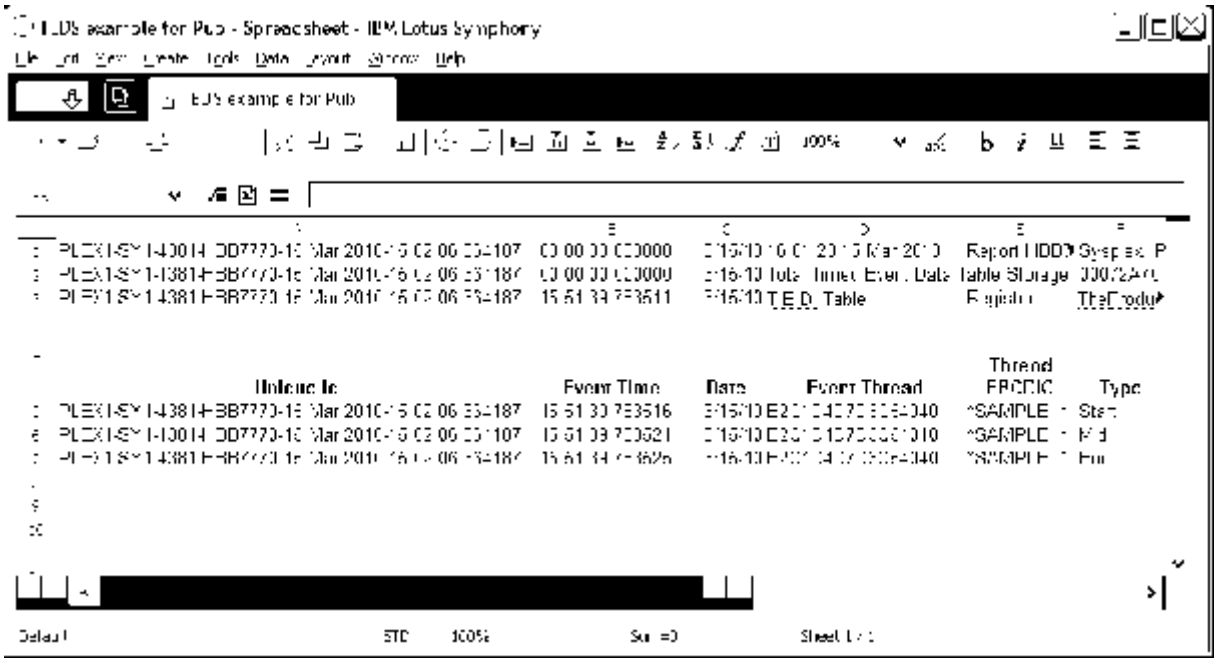


Figure 7. Sample (beginning portion) Timed Event Data spreadsheet

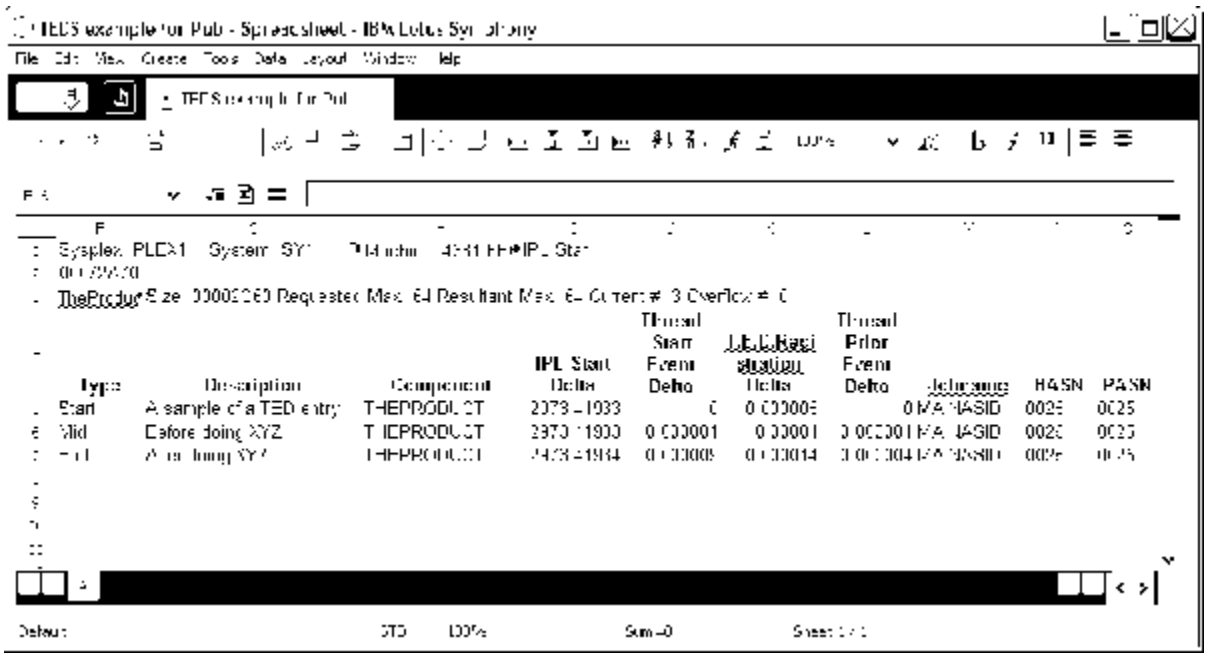


Figure 8. Sample (second portion) Timed Event Data spreadsheet

The formatted Timed Event Data Report will consist of:

- A header section for the start of the report.
- Timed Event Data Table sections with each Timed Event Data Table section comprised of a header section, the Timed Event Data Table formatted information, Timed Event Data Table formatted entries, and a trailer section.
- A header section for the IPST (IPL Statistics Table) followed by the formatted IPST.
- The spreadsheet format of the IPST and Timed Event Data entries.

The following describes each of the formatted sections in more detail:

The header section for the start of the report contains the text "IBM z/OS Timed Event Data Report" and the following fields:

- Level: The product and version of the Timed Event Data Report REXX exec (IEAVFTED).
- Date/Time of Report: The local date and time when the report was run.
- Component Filter: The filter used to select which entries to format. If no filter was specified, ALL is displayed.
- Sysplex: The name of the sysplex for the system from which the report was obtained.
- System: Name of the system from which the report was obtained.
- FMID: The FMID of the system from which the report was obtained and the z/OS release level.
- Machine: The model of the machine where the report was run.
- Online CPs: The number of online standard CPs, IBM zEnterprise® Application Assist Processors (zAAPs) and IBM z Integrated Information Processors (zIIPs).
- IPL Start Date/Time: The local date and time when the IPL was started for the system from which the report was obtained.
- Total Timed Event Data Table Storage: The total number of hexadecimal bytes of storage that is currently in use for the Timed Event Data Tables.

The header section for a Timed Event Data Table includes the following fields:

- Component: The value specified for the COMPNAME keyword on the IEATEDS REGISTER request.
- Address: The address in storage where the Timed Event Data Table resides.

- Table Size: Number of hexadecimal bytes allocated for the Timed Event Data Table.
- Register Date/Time: The local date and time when the Timed Event Data Table was registered with the IEATEDS REGISTER request.
- Requested MaxEvents: The maximum number of events originally specified on the IEATEDS REGISTER request.
- Resultant MaxEvents: The maximum number of events for this Timed Event Data Table that can be recorded with the IEATEDS RECORD request. This value may be the requested value, or a reduced value that allowed the Timed Event Data Table to be built within the 2M-byte storage limit.
- Current® NumEvents: The number of events that have been recorded thus far.
- Overflow NumEvents: The number of events that were not recorded because the Timed Event Data Table is full.

Following the Timed Event Data Table formatted header section, the report continues with zero or more formatted events that were recorded with IEATEDS RECORD requests, with each request having the following fields:

- EntryNum: This will start with 1 for each Timed Event Data Table and will increment for each event.
- Event Type/Thread: The type will be Start, Mid, or End as was specified for the EVENTTYPE keyword on the IEATEDS RECORD request. For Timed Event Data entries, the Thread, which is the value specified on the EVENTTREAD keyword of the IEATEDS RECORD request, will follow as formatted hex and again as formatted EBCDIC contained within asterisk borders. For IPST entries, the thread will always be in EBCDIC.
- Event Date/Time: The local date and time that the event was recorded.
- "*** Incomplete Event ***" will be displayed if it is determined that the entry is incomplete. In this case, the data for this event should be ignored.
- Description: The value specified for the DESCRIPTION keyword on the IEATEDS RECORD request.
- HASN: The ASID (address space identity) for the home address space at the time of IEATEDS RECORD request.
- PASN: The ASID (address space identity) for the primary address space at the time of IEATEDS RECORD request.
- Jobname: The jobname for the home address space at the time of the IEATEDS RECORD request.
- TCB: The TCB (Task Control Block) address at the time of IEATEDS RECORD request (which will be zero when running as an SRB).
- Module/Level/Offset: The values specified on the MODNAME and MODLEVEL keywords on the IEATEDS RECORD request, and the offset in the module where the IEATEDS RECORD request was issued. Note that the offset is calculated by obtaining the difference between the current location and SYSECT which names the current control section but which might not necessarily be the name of the module.
- SRB/Task Time: The SRB and task time values for the home address space at the time of IEATEDS RECORD request.
- User Data: The value(s) specified for the USERDATA keyword on the IEATEDS RECORD request, displayed as both printable hex and printable EBCDIC enclosed in asterisk borders. Note that zeros are appended to the user data to ensure that there are 16 bytes of data in the case where the specified user data combined size is less than 16 bytes.
- OUXBFCON: The formatted time from OUXBFCON which contains the accumulated I/O FICON® connect time for the address space.
- OUXBFDIS: The formatted time from OUXBFDIS which contains the accumulated I/O FICON disconnect time for the address space.
- OUXBFMNO: The formatted time from OUXBFMNO which contains the FICON magic number - for every I/O interrupt from a device attached to a FICON native CHPID, IOS will add one millisecond to this field.
- OUXBFWAIT: The formatted time from OUXBFWAIT which contains the accumulated I/O FICON wait time for the address space. This value includes pending time and control unit queue time.

- IPL Start Delta: The elapsed time from the start of the IPL to the time of the IEATEDS RECORD request for this event.
- Timed Event Data Registration Delta: The elapsed time from when the IEATEDS REGISTER request was made to the time of the IEATEDS RECORD request for this event.
- Thread Start Event Delta: The elapsed time from when the IEATEDS RECORD request with an EVENTTYPE of START was made to the time of the IEATEDS RECORD request for this event and this event has an EVENTTHREAD value that matches the START event.
- Thread Prior Event Delta: The elapsed time from when the IEATEDS RECORD request for the prior event in the same Timed Event Data Table with the same Event Thread was made to the time of the IEATEDS RECORD request for this event.

The trailer section for the Timed Event Data Table includes the following fields:

- Component: The value specified for the COMPONENT keyword on the IEATEDS REGISTER request.
- Number of Events: The number of processed START, MID, and END events. The number of start events and end events should normally match unless there were incomplete entries or an overflow of entries.

Note that error messages may be issued if a storage access error occurs, in which case the IPST, one of more Timed Event Data Tables, or Timed Event Data Table entries may be missing from the report.

Other error messages may be issued for data set errors or processing errors.

The Timed Event Data Report REXX exec provides the following return codes:

- Return Code=d'00' - Report written successfully.
- Return Code=d'16' - Report was not completed. An error message will be output to either the screen or within the report.

If you want to load the Timed Event Data Report into a spreadsheet program, perform the following:

- If the Timed Event Data Report was placed into a 512 character data set or a z/OS UNIX file, edit the data set (or a copy) to delete everything above the spreadsheet data (including the directions).
- If the Timed Event Data Report was placed into a 255 character data set (i.e., obtained from a dump under IPCS), invoke IEAVFTED with the IPCSDA option which will extract the spreadsheet data from the 255 character data set and place it into either a pre-allocated 512 character data set or a z/OS UNIX file.
- When you download or FTP the data set, choose the download options of ASCII or TEXT. Do not download as binary.
- When importing into the spreadsheet program, indicate that the data is delimited with a delimiter of a semicolon (or the character that was specified with the SS keyword when IEAVFTED was invoked to generate the Timed Event Data Report).
- Sorting by the 'Event Time' column will put all the data into chronological order.
- Sorting by the 'Event Thread' and the 'Event Time' columns will group all the related events in chronological order. Examining the 'Thread Prior Event Delta' will help identify events that took a long time.

Chapter 55. IEATXDC — Transactional execution diagnostic controls

Description

In support of the diagnostic controls of transactional execution, as defined in the *z/Architecture Principles of Operation*, the following services are provided:

For the current task,

- Indicate the scope of the diagnostic controls.
- Set the diagnostic controls for "no abort".
- Set the diagnostic controls for "abort every".
- Set the diagnostic controls for "abort random".

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with PSW key 8-15. SCOPE=ALL requires supervisor state; if a problem state caller indicates SCOPE=ALL, it is treated as SCOPE=PROBLEM.
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold any lock(s). No locks are required.
Control parameters:	None.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IEATXDC macro, the caller does not have to place any information into any general purpose register (GPR).

Before issuing the IEATXDC macro, the caller does not have to place any information into any access register (AR).

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The IEATXDC macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEATXDC.
IEATXDC	
ƀ	One or more blanks must follow IEATXDC.
SCOPE=PROBLEM	
SCOPE=ALL	
,OPERATION=NO_ABORT	
,OPERATION=SET_EVERY	

Syntax	Description
,OPERATION=SET_RANDOM	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEATXDC macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

SCOPE=PROBLEM
SCOPE=ALL
A required parameter that identifies the scope of the diagnostic controls.

SCOPE=PROBLEM
indicates that the diagnostic controls apply only for problem state transactional execution.

SCOPE=ALL
indicates that the diagnostic controls apply both to problem state and supervisor state transactional execution. If a problem state caller requests SCOPE=ALL, however, it is treated as SCOPE=PROBLEM.

,OPERATION=NO_ABORT
,OPERATION=SET_EVERY
,OPERATION=SET_RANDOM
A required parameter that identifies the type of operation to perform.

,OPERATION=NO_ABORT
indicates to set the transactional diagnostic controls for this task so that the system will not apply its SET_EVERY or SET_RANDOM rules. Transactions themselves may still abort for all the defined architectural reasons.

,OPERATION=SET_EVERY
indicates to set the transactional diagnostic controls for this task to request abort of every nonconstrained transaction.

,OPERATION=SET_RANDOM
indicates to set the transactional diagnostic controls for this task to request abort of random nonconstrained transactions.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

ABEND codes

None.

Return codes

When the IEATXDC macro returns control to your program, GPR 15 (and *retcode*, when you code RETCODE) contains a return code.

The following table identifies the hexadecimal return and reason codes.

Table 62. Return codes for the IEATXDC Macro

Return Code	Meaning and Action
0	**Meaning**: Successful completion. Diagnostic controls are set to the requested value **Action**: None required.
4	**Meaning**: Warning. The machine does not support transactional execution. Diagnostic controls are not set. **Action**: Avoid calling IEATXDC when the machine does not support transactional execution.
8	**Meaning**: Unexpected input. **Action**: Check for possible storage overlay.
12	**Meaning**: Service called in SRB mode. **Action**: Avoid using IEATXDC in SRB mode.

Examples

None.

Chapter 56. IEAVAPE — Allocate_Pause_Element

Description

Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• For auth_level=IEA_UNAUTHORIZED: Task • For auth_level=IEA_AUTHORIZED: Task or SRB
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	• When supervisor state and PSW key 0: The local lock may be held. • When problem state, or not PSW key 0: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Only 2040 unauthorized PETs may be allocated at any one time in an address space.

Input register information

Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return Code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVAPE	(return_code ,auth_level ,pause_element_token)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants defined in IEAASM or IEAC, as appropriate. The level desired results from adding the values of the required types together. The authorization type is not optional.

For instance, the level to allocate authorized pause elements that are checkpoint/restart tolerant is IEA_AUTHORIZED + IEA_CHECKPOINTOK, or 3.

The following levels are supported:

Table 63. Authorization

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	When using the allocated pause element through other services, either auth_level IEA_UNAUTHORIZED or IEA_AUTHORIZED can be used.
IEA_AUTHORIZED	1	When using the allocated pause element through other services, auth_level=IEA_AUTHORIZED will be required. Caller must be both key 0 and supervisor state.

Table 64. Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_CHECKPOINTOK	2	The application can tolerate the pause elements' not being restored upon a restart after a checkpoint.

Note: If the IEA_CHECKPOINTOK value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.

,pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element which you can use to synchronize the processing of a task.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (0) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
24 (18) IEA_LOCK_HELD	**Meaning:** Program error. If the auth_level indicates AUTHORIZED, locks other than the local lock are held. If the auth_level indicates UNAUTHORIZED, locks are held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
56 (38) IEA_NO_PETS_AVAILABLE	**Meaning**: There are no pause element tokens available. **Action**: Retry the request later.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
68 (44) IEA_XFER_TO_SELF	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
72 (48) IEA_XFER_FAILED	**Meaning**: Environmental error. The system could not obtain storage for a pause element. The system rejects the service call. **Action**: Retry the request later. If the problem persists, consult your system programmer.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 57. IEAVAPE2 — Allocate_Pause_Element

Description

Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	• For LINKAGE=BRANCH: The local lock may be held. • For LINKAGE=SVC: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Only 2040 unauthorized PETs may be allocated at any one time in an address space.

Allocate_Pause_Element cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC and pause_element_owner_stoken as binary zero.

Input register information

Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVAPE2	,(return_code ,pause_element_auth_level ,pause_element_token ,pause_element_owner_stoken ,owner_termination_release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

,pause_element_auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants defined in IEAASM or IEAC, as appropriate. The level desired results from adding the values of the required types together. The authorization type is not optional.

For instance, the level to allocate authorized pause elements that are checkpoint/restart tolerant is IEA_AUTHORIZED + IEA_CHECKPOINTOK, or 3.

The following levels are supported:

Table 65. Authorization

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	When using the allocated pause element through other services, either pause_element_auth_level IEA_UNAUTHORIZED or IEA_AUTHORIZED can be used.
IEA_AUTHORIZED	1	When using the allocated pause element through other services, pause_element_auth_level =IEA_AUTHORIZED is required. Caller must be both key 0 and supervisor state.

Table 66. Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_CHECKPOINTOK	2	The application can tolerate the pause elements' not being restored upon a restart after a checkpoint.

Note: If the IEA_CHECKPOINTOK value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.

,pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies a pause element which you can use to synchronize the processing of a task or SRB.

,pause_element_owner_stoken
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

Specifies the space token (STOKEN) of the address space which is to be considered the owner of the Pause Element being allocated. Specify one of the following values:

- Binary zero: indicate the system should make the current primary address space the owner of the Pause Element. This is the only value valid for key 8-15 problem state callers.
- A valid STOKEN, indicate the system should make the address space with the matching STOKEN the owner for the pause element.

When the CMRO task (the first job step task) of an address space terminates, the system will release and deallocate any pause elements owned by the CMRO task's home address space. The table below describes exactly when the system will release and/or deallocate a Pause Element:

Allocation Service version:	**Deallocation Rules**
IEAVAPE	The PE will be deallocated by the system when one of the following events occurs: • The PE was never used to pause a task or SRB and the CMRO task for the space which allocated it terminates. • The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ • The CMRO task of the home address space of the task or SRB which last used the PE terminates and the PE is not being used to pause an SRB. The home address space of the task or SRB which last used the PE terminates.

Allocation Service version:	Deallocation Rules
IEAVAPE2	The PE will be deallocated by the system when one of the following events occurs: • The CMRO task of the address space specified by pause_element_owner_stoken terminates. If the PE is being used to pause a DU when the CMRO task terminates, the system will release the DU using the owner_termination_release_code before the PE is deallocated. Note that in this case, the UPET returned will be 16 bytes of binary zeros, an invalid value. • The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ • The PE is being used to pause a task or SRB when the home address space of the task or SRB is terminated • The CMRO task of the address space which owns the PE terminates and the PE is not being used to pause an SRB. The address space which owns the PE terminates. Note: A PE is considered as "being used to pause a task or SRB," when the PE is not Reset or Prereleased.

,owner_termination_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Specifies the release code which will be returned to a paused DU if the system deallocates the pause element while it is being used to pause a task or SRB, due to the CMRO task of its owning address space terminating.

Note: If the system deallocates a PE due to its owner terminating while the PE was not being used to pause a task or SRB, future attempts to use the PE will fail with a return code indicating the PETOKEN was stale or the PE is in an invalid state.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Allocate_Pause_Element service routine is to be invoked. The following options are supported:

Table 67. Linkage option

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Allocate_Pause_Element service routine will be invoked via an SVC linkage. This option can be used when in non-cross memory task mode, any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Allocate_Pause_Element service routine will be invoked via a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
00 (0)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. One or more locks other than the local lock are held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
40 (28)	IEA_INVALID_AUTHCODE	**Meaning:** Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
48 (30)	IEA_OUT_OF_STORAGE	**Meaning:** Environmental error. The system could not obtain storage for a pause element. The system rejects the service call. **Action:** Retry the request later. If the problem persists, consult your system programmer.
56 (38)	IEA_NO_PETS_AVAILABLE	**Meaning:** There are no pause element tokens available. **Action:** Retry the request later.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Contact IBM support.
84 (54)	IEA_INVALID_LINKAGE	**Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
88 (58)	IEA_INVALID_OWNER_STOKEN	**Meaning:** Program error. The stoken specified for pause_element_owner_stoken is not valid. **Action:** Obtain the correct stoken of the target and reissue the call
96 (60)	IEA_UNAUTH_NONZERO_OWNER_STOKEN	**Meaning:** Program error. A key 8-15 problem state caller specified a nonzero value for pause_element_owner_stoken **Action:** Check the calliing program for a probable coding error. Correct the program and rerun it.
100 (64)	IEA_INVALID_AUTHLVL_AUTHCODE	**Meaning:** The pause_element_auth_level value specified in the call is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.

Chapter 58. IEAVDPE — Deallocate_Pause_Element

Description

Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• For auth_level=IEA_UNAUTHORIZED: Task • For auth_level=IEA_AUTHORIZED: Task or SRB
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	• When supervisor state and PSW key 0: The local lock may be held. • When problem state, or not PSW key 0: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL IEAVDPE	,(return_code ,auth_level ,pause_element_token)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	This pause element being deallocated must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	This pause element being deallocated must have been allocated with auth_level=IEA_AUTHORIZED.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (00) IEA_SUCCESS	**Meaning**: Successful completion **Action**: None.
04 (04)	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20) IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the specified pause element token is not valid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 59. IEAVDPE2 — Deallocate_Pause_Element

Description

Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	• For LINKAGE=BRANCH: The local lock may be held. • For LINKAGE=SVC: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Input register information

Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL IEAVDPE2	,(return_code ,pause_element_token ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Deallocate_Pause_Element service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Deallocate_Pause_Element service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Deallocate_Pause_Element service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning:** Successful completion **Action:** None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. One or more locks other than the local lock are held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning:** Program error. The pause element associated with the specified pause element token is invalid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for an unauthorized pause element allocated to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
84 (54)	IEA_INVALID_LINKAGE	**Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Contact IBM support.

Chapter 60. IEAVPME2 — Pause multiple elements service

Description

IEAVPME2 is a callable service that can be used to pause on one or more pause element tokens (PETs). When the specified number of pause elements (PEs) represented by PETs has been released, you receive control back with the following:

- A list of PETs that you can use to pause on again
- An indication of which PEs were released
- Their release codes.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – If any input PET was allocated as IEA_AUTHORIZED, supervisor state and PSW key 0. – If all input PETs were allocated as IEA_UNAUTHORIZED, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=SASN=HASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC. IEA_LINKAGE_SVC is limited to pausing upon no more than 1000 PETs.

Pause cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the EXEC PGM=xxx task).

Input register information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Note that register 13 is not required to contain any particular value. See the **workarea** parameter description.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code.

Note that this service saves and restores full 64-bit GPRs.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

There is a maximum number of PETs which can be processed very quickly by IEAVPME2 without doing additional GETMAINs and FREEMAINs. That number is currently 16.

Syntax

Syntax	Description
CALL IEAVPME2	(return_code ,pause_element_token_list ,updated_pause_element_token_list ,release_code_list ,number_of_PETs_in_each_list ,number_of_PEs_to_release ,linkage ,workarea)

Link-edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEAVPME2:

1.
```
LOAD EP=IEAVPME2
Save the entry point address...
Put the saved entry point address into R15
CALL (15),(...)
```

2.
```
L     15,X'10'(0,0)
L     15,X'220'(15,0)
L     15,X'14'(15,0)
L     15,X'100'(15,0)
CALL  (15),(...)
```

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the highest return code from the Pause service (multiple return codes are possible when more than one PET has been specified – see*release_code_list*). When the low-order bit of the return code is on, *release_code_list* contains the return codes for individual PETs rather than release codes.

Note that no pause has actually occurred if the return code is non-zero. In this situation, any PETs which have been released remain released.

,pause_element_token_list
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes times the number of PEs you want to pause on

A list of the PETs identifying the PEs you want to pause on. *Number_of_PETs_in_each_list* specifies how many PETs are in the list.

,updated_pause_element_token_list
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes times the number of PEs you want to pause on

If *return_code* is 0, a list of the PETs returned by Pause Multiple Elements. Each entry corresponds to an entry in the *pause_element_token_list*. For each PE that was released, the system puts an updated PET into this list. For PEs that are not released, the entry contains the PET from the original *pause_element_token_list*. These new PETs must be used in place of the PETs specified in *pause_element_token_list* or *pause_element_token* on future calls to the Pause, Release, Transfer, or Deallocate_Pause_Element service. The first byte of each entry in *release_code_list* identifies which PEs were released. *Number_of_PETs_in_each_list* specifies how many PETs are in the list.

If *return_code* is not 0, the PETs are not updated and this list is not returned.

If the paused *workunit* was released by the system (the release code is the *owner_termination_release_code* specified on the IEAVAPE2 allocation), the PET returned in that slot will be 16 bytes of binary zeros, an invalid value.

,release_code_list
Returned parameter

- Type: Fullword
- Character Set: N/A
- Length: 4 bytes times the number of PEs you want to pause on

Each entry corresponds to an entry in the *pause_element_token_list*.

If *return_code* is 0, the pause was successful and has been released. For each PE that was Released, the system puts X'01' into the first byte and the release code for that PET into the second through fourth bytes of the full-word entry for that PET. For PEs which are not released, the entry contains binary zeroes.

If *return_code* is 5, 9, D, 21, 35, 3D, or 41, a problem was found with at least one of the PETs specified in *pause_element_token_list*and the pause request did not complete.

The individual return code for each PET is in the second through fourth bytes of the *release_code_list* entry for that PET. Each PET with a return code of 0 would have been paused on if all the other PETs in the list had received return codes of 0. If *return_code* contains any other value, the pause request did not complete and *release_code_list* does not contain any meaningful information.

If *return_code* contains any other value, the pause request did not complete and *release_code_list* does not contain any meaningful information.

Note: These return codes are the equivalent of return codes 4, 8, C, 20, 34, 3C, and 40 for IEAVPSE2 (Pause single), with the addition of a low-order bit to signify that the *release_code_list*contains individual PET return codes.

,number_of_PETs_in_each_list
Supplied parameter

Number_of_PETs_in_each_list specifies how many entries are in *release_code_list*.

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

This number specifies how many PETs you want to Pause on. This number also specifies the number of entries in *pause_element_token_list*, *updated_pause_element_token_list* , and *release_code_list*. For SVC entry callers, the maximum number that can be specified is 1000.

,number_of_PEs_to_release
Supplied parameter

Number_of_PETs_in_each_list specifies how many entries are in *release_code_list*.

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

This number specifies how many PEs must be released before control is returned back to the issuer of Pause Multiple Elements. This number must be at least 1 and no more than *number_of_PETs_in_each_list*.

Note that if more PEs than this number were released before IEAVPME2 was issued (pre-released), the number of updated PETs in *updated_pause_element_token_list* will be the actual number released.

,linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Pause service routine is to be invoked. The following options are supported:

Table 68. Linkages

Variable	Value	Meaning
IEA_LINKAGE_SVC	0	The Pause service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Pause service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

,workarea
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 216 bytes

Specifies a work area on a doubleword boundary in which the Pause service routine will save information about the caller including the caller's registers. This can be the same area that R13 points to if the first 216 bytes of that area can be used as an F7SA savearea.

ABEND codes

None. However, note that you may receive a GETMAIN abend if this service is unable to obtain storage needed to process the PETs.

Return codes

When IEAVPME2 returns control to your program, if GPR 15 contains 0, 5, 9, D, 21, 35, 3D, or 41, *release_code_list* contains information about the status of each PET that was specified. If GPR 15 contains any other value, *release_code_list*does not contain any meaningful information.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
00 (0)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None.
05 (05)	IEA_PE_TOKEN_BAD	**Meaning:**Program error. The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
09 (09)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
13 (0D)	IEA_DUPLICATE PAUSE	**Meaning:** The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. The caller is holding one or more locks; no locks may be held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
33 (21)	IEA_PE_BAD_STATE	**Meaning:** Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action:** Check the calling program for a probable coding error, such as attempting to perform a pause or transfer using a pause element token that has already been used to pause or transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_ RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
53 (35)	IEA_ALREADY_SUSPENDED	**Meaning:** The pause element was already paused. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
61 (3D)	IEA_AUTH_LEVEL_MISMATCH	**Meaning:** Program error. The caller was in problem state or key 8, but the pause element token was allocated with *pause_element_auth_level*=IEA_AUTHORIZED. The system rejects the service call. **Action:** Program error. The specified pause element token is not valid. The system rejects the service call.
65 (41)	IEA_PE_NOT_HOME_PM	**Meaning:** Program error. The pause element token was for a pause element allocated with *pause_element_auth_level*=IEA_AUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
76 (4C)	IEA_ABENDED_47B	**Meaning:** After an SRB received abend 47B, it invoked IEAVPME2. It is not valid to invoke IEAVPME2 after receiving abend 47B. **Action:** Update the calling program to not invoke IEAVPME2 after abend 47B.
80 (50)	IEA_INSUSPEND_EXIT	**Meaning:** The suspend exit specified on SUSPEND with SPTOKEN of an SRB invoked IEAVPME2. It is not valid to invoke IEAVPME2 from a suspend exit. **Action:** Update the calling program to not invoke IEAVPSE from a suspend exit.
84 (54)	IEA_INVALID_LINKAGE	**Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
104 (68)	IEA_INVALID_NUMBER_OF_PETS	**Meaning:** Program error. The number of PETs specified for Pause Multiple was 0 or for SVC entry users more than 1000. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
108 (6C)	IEA_INVALID_NUMBER_TO_ RELEASE	**Meaning:** Program error. The number of PEs to release is 0 or greater than the number of PETs specified for Pause Multiple. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4094 (FFE)	IEA_ERROR_PETS_INVALIDATED	**Meaning:** Pause processing has encountered an error and cannot complete the Pause request. This input PETs have been invalidated. This return code is only issued for Pause Multiple requests. **Action:** Do not return the PETs to the system using the Deallocate Pause Elements services. Note that new PEs must be obtained before attempting to pause again.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Contact IBM support.

Chapter 61. IEAVPSE — Pause service

Description

Call Pause to make the current task nondispatchable. Once you pause a task, it remains nondispatchable until a Release service specifying the same PET is called. That is, the program issuing the Pause does not receive control back until after the Release occurs.

If a Release service specifying the same PET is called before Pause, the system returns control immediately to the calling program, and the task is not paused.

When you use Pause, it returns an updated PET; you use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program is running auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only pause another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVPSE	,(return_code ,auth_level ,pause_element_token ,updated_pause_element_token ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Pause service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum level that the specified pause element was allocated with. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	The pause element being paused must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	The pause element being paused must have been allocated with auth_level=IEA_AUTHORIZED.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element being used to pause the current task. You obtain the PET from the Allocate_Pause_Element service.

Once you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to Pause the task; use the new PET the next time you make a Pause request using the same Pause element.

,updated_pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token, which cannot be reused after a successful call to Pause.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A

- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task from its paused condition.

ABEND codes

Abend Code	Reason Code	Description
AC7	001A0001	This is an internal error. Contact IBM support.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (00) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C) IEA_DUPLICATE_PAUSE	**Meaning**: The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20) IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
52 (34) IEA_ALREADY_SUSPENDED	**Meaning**: The pause element was already paused. **Action**: Check the calling program for a probable coding error and correct the program and rerun it.
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 62. IEAVPSE2 — Pause service

Description

Call Pause to make the current task or SRB nondispatchable. When you pause a task or SRB, it remains nondispatchable until a Release or Transfer specifying the same PET is called. That is, the program issuing the Pause does not receive control back until after the Release or Transfer occurs. At that time, the returned release_code will contain a value supplied by the associated Release or Transfer request.

If a Release service specifying the same PET is called before Pause, the system returns control immediately to the calling program, and the task or SRB is not paused.

When you use Pause, it returns an updated PET; you use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Pause cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVPSE2	,(return_code ,pause_element_token ,updated_pause_element_token ,release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Pause service.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element being used to pause the current task or SRB. You obtain the PET from the Allocate_Pause_Element service.

When you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task or SRB; use the new PET the next time you make a Pause request using the same Pause element.

,updated_pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token. This new PET must be used in place of the PET specified in pause_element_token on future calls to the Pause, Release, Transfer, or Deallocate_Pause_Element service. If the paused workunit was released by the system (the release code is the owner_termination_release_code specified on the IEAVAPE1 allocation), the PET returned will be 16 bytes of binary zeros, an invalid value.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Pause service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Pause service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Pause service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

Abend Code	Reason Code	Description
AC7	001A0001	This is an internal error. Contact IBM support.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None
04 (04)	IEA_PE_TOKEN_BAD	**Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C)	IEA_DUPLICATE_PAUSE	**Meaning:** The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
32 (20)	IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
52 (34)	IEA_ALREADY_SUSPENDED	**Meaning:** The pause element was already paused. **Action:** Check the calling program for a probable coding error and correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action:** Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error and correct the program and rerun it.
76 (4C)	IEA_ABENDED_47B	**Meaning:** After an SRB received ABEND 47B, it invoked IEAVPSE2. It is not valid to invoke IEAVPSE2 after receiving ABEND 47B. **Action:** Update the calling program to not invoke IEAVPSE2 after ABEND 47B.
80 (50)	IEA_IN_SUSPEND_EXIT	**Meaning:**The suspend exit specified on SUSPEND with SPTOKEN of an SRB invoked IEAVPSE2. It is not valid to invoke IEAVPSE2 from a suspend exit. **Action:** Update the calling program to not invoke IEAVPSE2 from a suspend exit.
84 (54)	IEA_INVALID_LINKAGE	**Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Contact IBM support.

Chapter 63. IEAVRLS — Release

Description

Call Release to remove a task that has been paused, or to keep a task from being paused. Although a pause element can be used multiple times to pause a task, a pause element token can be used to successfully pause and release a task only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the Pause and Transfer services.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	May hold the CPU, local or CMS lock.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVRLS	,(return_code ,auth_level ,target_du_pause_element_token ,target_du_release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return from the Release service.

,auth_level
Supplied Parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level that the specified pause element was allocated with. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	The pause element being released must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	The pause element being released must have been allocated with auth_level=IEA_AUTHORIZED.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element used to pause the task. If the PET identifies a pause element that has not been paused (that is, the task has not been paused), the task will not be paused. However, the value specified in target_du_release_code will be returned to the caller of Pause.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of Pause or Transfer service that used (or will use) the same PET to pause a task. If your program is not using this code for communication, set this field to zero.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (00) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04) IEA_PE_TOKEN_BAD	**Meaning**: The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10) IEA_SLEEP_DISRUPTED	**Meaning**: RTM has terminated the task; no release is necessary. **Action**: None
20 (14) IEA_SPACE_TERMINATING	**Meaning**: The address space that contains the task that is terminating; no release is necessary. **Action**: None
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one o r more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20) IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified is invalid or has already been prereleased. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 64. IEAVRLS2 — Release

Description

Call Release to remove a task or SRB that has been paused, or to keep a task or SRB from being paused.

Although a pause element can be used multiple times to pause a task or SRB, a pause element token can be used to successfully pause and release a task or SRB only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the Pause and Transfer services.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	• For LINKAGE=BRANCH: The CPU, CMS, or local locks may be held. • For LINKAGE=SVC: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Release cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVRLS2	,(return_code ,,target_du_pause_element_token ,target_du_release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return from the Release service.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element used to pause a task or SRB. You obtain the PET from the Allocate_Pause_Element service.

When you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task or SRB; use the new PET the next time you make a Pause request using the same Pause element.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of Pause or Transfer service that used (or will use) the same PET to pause a task or SRB. If your program is not using this code for communication, set this field to zero.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Release service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Release service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Release service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning:** The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10)	IEA_SLEEP_DISRUPTED	**Meaning:** RTM has terminated the task or SRB; no release is necessary. **Action:** None
20 (14)	IEA_SPACE_TERMINATING	**Meaning:** The address space that contains the task or SRB is terminating; no release is necessary. **Action:** None
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. The caller is holding one or more locks; other than the local lock, CMS, or CPU lock, no locks may be held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning:** Program error. The pause element associated with the pause element token specified is invalid or has already been prereleased. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.

Return code: Decimal (Hex)	Equate symbol	Meaning and Action
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action:** Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
84 (54)	IEA_INVALID_LINKAGE	**Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Contact IBM support.

Chapter 65. IEAVRPI — Retrieve_Pause_Element_Information service

Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- Its authorization level
- The address space that currently owns it
- Its current state (Reset, Prereleased, Paused, or Released)
- If its state is Prereleased or Released, its Release Code

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program can do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB • When problem state, or not PSW key 0: Task
Cross memory mode:	• When supervisor state and PSW key 0: Any PASN, any HASN, any SASN • When problem state, or not PSW key 0: PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

None.

Input register information

Before calling the Retrieve_Pause_Element_Information service, the caller does not need to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVRPI	,(return_code ,auth_level ,pause_element_token ,authorization ,owner ,state ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Retrieve_Pause_Element_Information service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the caller's authorization level. The following levels are supported: IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which can be used by the calling program.

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The caller is not key 0 and supervisor state.
IEA_AUTHORIZED	1	The caller is both key 0 and supervisor state.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information will be returned. You obtain the PET from the Allocate_Pause_Element service.

,authorization
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The authorization level of the creator of the pause element specified by the input PET.

One of the following values:

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The caller is not key 0 and supervisor state.
IEA_AUTHORIZED	1	The caller is not key 0 and supervisor state.
IEA_UNAUTHORIZED + IEA_CHECKPOINTOK	2	Unauthorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.
IEA_AUTHORIZED + IEA_CHECKPOINTOK	3	Authorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.

,owner
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

The Stoken of the address space that currently owns the pause element specified by the input PET.

,state
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state may have changed after it was obtained.

State Constant Hexadecimal (Decimal)	**Meaning**
IEAV_PET_PRERELEASED 1 (1)	The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.
IEAV_PET_RESET 2 (2)	The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.
IEAV_PET_RELEASED 40 (64)	The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transitioned the PE into the RESET state.
IEAV_PET_PAUSED 80 (128)	A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

Note: The returned value is random if the state parameter is not IEAV_PET_RELEASED or IEAV_PET_PRERELEASED.

ABEND codes

None.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified an unauthorized auth_level type, but a pause element token allocated with an authorized auth_level type was encountered. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified an unauthorized auth_level type, but a pause element token for a pause element allocated to another address space was specified. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 66. IEAVRPI2 — Retrieve_Pause_Element_Information service

Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- Its authorization level
- Its current state (Reset, Prereleased, Paused, or Released)
- If its state is Prereleased or Released, its Release Code
- The stoken of the owner of the pause element (See IEAVAPE — Allocate_Pause_Element for details of ownership).
- The stoken of the home address space of the task or SRB which is paused by the pause element.

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program may do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Key 2-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Input register information

Before calling the Retrieve_Pause_Element_Information service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list

13
Address of a 72-byte register save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVRPI2	,(return_code ,pause_element_auth_level ,pause_element_token ,linkage ,owner_stoken ,current_stoken ,state ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Retrieve_Pause_Element_Information service.

,pause_element_auth_level
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the authorization level with which the pause element specified by the input PET was allocated. The following levels are supported:

Variable	Value (hexadecimal)	Meaning
IEA_PET_UNAUTHORIZED	0	The pause element was allocated with pause_element_auth_level=IEA_UNAUTHORIZED.
IEA__PET_AUTHORIZED	1	The pause element was allocated with pause_element_auth_level=IEA_AUTHORIZED.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information will be returned. You obtain the PET from the Allocate_Pause_Element service.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A

- Length: 4 bytes

The calling program can use the constants defined in IEAASM or IEAC, as appropriate. Add the specified values together to achieve the desired results. For example, to specify linkage branch and untrusted PET, specify IEA_LINKAGE_BRANCH + IEA_UNTRUSTED_PET.

The following options are supported:

Table 69. Linkage variables

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Retrieve_Pause_Element_Information service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Retrieve_Pause_Element_Information service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

Table 70. Untrusted attribute linkage variable

Variable	Value (hexadecimal)	Meaning
IEA_UNTRUSTED_PET	2	The Retrieve_Pause_Element_Information service routine is to validate that the input PET is allowed to be used by an unauthorized program. An authorized program should use this if it is validating a PET provided to it by an unauthorized caller.

,owner_stoken
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

Specifies the stoken of the address space that currently owns the pause element specified by the input PET. The owner of a PE allocated by IEAVAPE2 is static and specified on the IAVEAPE2 call. The owner of a PE allocated by IEAVAPE is dynamic. See IEAVAPE — Allocate_Pause_Element for details.

,current_stoken
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

If the value returned in state is IEA_PET_PAUSED, The stoken of the home address space of the task or SRB which is paused by the specified pause element. If the value in state is not IEA_PET_PAUSED, the information returned in this parameter is undefined.

,state
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state may have changed after it was obtained.

State Constant Hexadecimal (Decimal)	Meaning
IEAV_PET_PRERELEASED 1 (1)	The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.
IEAV_PET_RESET 2 (2)	The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.
IEAV_PET_RELEASED 40 (64)	The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transitioned the PE into the RESET state.
IEAV_PET_PAUSED 80 (128)	A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

Note: The returned value is random if the state parameter is not IEAV_PET_RELEASED or IEAV_PET_PRERELEASED.

ABEND codes

None.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None

Return code: Decimal (Hex)	Equate symbol	Meaning and Action
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28)	IEA_INVALID_AUTHCODE	**Meaning:** Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C)	IEA_AUTH_LEVEL_MISMATCH	**Meaning:** Program error. The caller was in Problem state or key 8, or specified IEA_UNTRUSTED_PET, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
84 (54)	IEA_INVALID_LINKAGE	**Meaning**: Program error. The linkage value specified is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 67. IEAVTPE — Test_Pause_Element service

Description

Call Test_Pause_Element to test a pause element and determine its state. If its state is Prereleased or Released, the pause element's release code will also be returned.

To ensure minimal overhead when you use the service, Test_Pause_Element establishes no recovery. You are responsible for supplying any needed recovery to handle errors that occur due to invalid input pause element Tokens or call state errors.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB • When problem state, or not PSW key 0: Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

None.

Input register information

Before calling the Test_Pause_Element service, the caller does not have to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVTPE	,(return_code ,pause_element_token ,state ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Test_Pause_Element service.

,pause_element_token
Supplied parameter

- Type: Character string

- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information is to be returned. You obtain the PET from the Allocate_Pause_Element service.

,state
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state may have changed after it was obtained.

State Constant Hexadecimal (Decimal)	**Meaning**
IEAV_PET_PRERELEASED 1 (1)	The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.
IEAV_PET_RESET 2 (2)	The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.
IEAV_PET_RELEASED 40 (64)	The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transitioned the PE into the RESET state.
IEAV_PET_PAUSED 80 (128)	A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task or SRB from its paused condition.

Note: The returned value is random if the state parameter is not IEAV_PET_RELEASED or IEAV_PET_PRERELEASED.

ABEND codes

None.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Chapter 68. IEAVXFR — Transfer service

Description

Call the Transfer service to release a paused task, and, when possible, give it immediate control. This service can also, optionally, pause the task under which the Transfer request is made. If the caller does not request that its task be paused, the caller's task remains dispatchable.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	When supervisor state and PSW key 0 and a current_du_pause_element_token of 16 bytes of binary zeros are specified, the local lock may be held. Otherwise, no locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only transfer to another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVXFR	,(return_code ,auth_level ,current_du_pause_element_token ,updated_pause_element_token ,current_du_release_code ,target_du_pause_element_token ,target_du_release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Transfer service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	The pause elements must have been allocated with auth_level=_UNAUTHORIZED.
IEA_AUTHORIZED	1	The pause elements must have been allocated with auth_level=_AUTHORIZED.

,current_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element used to pause the current task. Once a PET is used on a call to the Pause service, it cannot be reused on a second call to Pause or as a current_du_pause_element_token on Transfer. A new PET is returned to updated_pause_element_token. The new PET now properly defines the pause element and should be used the next time a pause, transfer, release, or deallocate_pause_element request is made using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task will not be paused. The updated_pause_element_token and current_du_release_code will be unpredictable.

CAUTION: Do not specify the same PET for both current_du_pause_element_token and target_pause_element_token.

,updated_pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to Pause or Transfer.

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

,current_du_release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the Release or Transfer service that released the current task from its paused condition.

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element to release the target task. Any PET that specifies a pause element not currently being used to pause a task is valid. When a PET for a previously released pause element is used to try to pause a task, the task is not paused; however, the value specified in target_du_release_code will still be returned to the caller of Pause or Transfer.

If the task was paused and is now dispatchable, the task will immediately be given control on the current processor.

CAUTION: Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the issuer of the Pause or Transfer service that is used (or will use) the same PET to pause a task.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C)	IEA_DUPLICATE_PAUSE	**Meaning**: The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10)	IEA_SLEEP_DISRUPTED	**Meaning**: RTM has terminated the task or SRB; no release is necessary. **Action**: None
20 (14)	IEA_SPACE_TERMINATING	**Meaning**: The address space that contains the task or SRB is terminating; no release is necessary. **Action**: None
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28)	IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
52 (34)	IEA_ALREADY_SUSPENDED	**Meaning**: The pause element was already paused. **Action**: Check the calling program for a probable coding error and correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
68 (44)	IEA_XFER_TO_SELF	**Meaning:** Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
72 (48)	IEA_XFER_FAILED	**Meaning**: The transfer failed, and the current_du_pause_element_token is no longer useable. **Action**: Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 69. IEAVXFR2 — Transfer service

Description

Call the Transfer service to release a paused task or SRB, and, when possible, give it immediate control. This service can also, optionally, pause the task or SRB under which the Transfer request is made. If the caller does not request that its task or SRB be paused, the caller's task or SRB remains dispatchable.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	If LINKAGE=BRANCH and a current_du_pause_element_token of 16 bytes of binary zeros is specified, the local lock may be held. Otherwise, no locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEACSS from SYS1.CSSLIB) or have the calling program LOAD and then CALL the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC.

Transfer cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
CALL IEAVXFR2	,(return_code ,current_du_pause_element_token ,updated_pause_element_token ,current_du_release_code ,target_du_pause_element_token ,target_du_release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Transfer service.

,current_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element that is being or will be used to pause a task or SRB. When a PET is used on a call to the Pause service, it cannot be reused on a second call to Pause or as a current_du_pause_element_token on Transfer. A new PET is returned to update_pause_element_token. The new PET properly defines the pause element and should be used the next time a pause, transfer, release, or deallocate_pause_element request is made using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task or SRB is not paused. The updated_pause_element_token and current_du_release_code will be unpredictable.

CAUTION: Do not specify the same PET for both current_du_pause_element_token and target_du_pause_element_token.

,updated_pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to Pause or Transfer.

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

,current_du_release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the Release or Transfer service that released the current task or SRB from its paused condition.

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies a pause element that is being or will be used to pause a task or SRB. If the task or SRB is paused, it will be released, and, if possible, be given control. If the task or SRB is not paused using the specified pause element, it will not be paused when an attempt to pause is made. In either case the task or SRB will be returned the value specified in target_du_release_code.

CAUTION: Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the issuer of the Pause or Transfer service used (or will use) the PET specified in target_du_pause_element_token to pause a task or SRB.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Transfer service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Transfer service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Transfer service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C)	IEA_DUPLICATE_PAUSE	**Meaning**: The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10)	IEA_SLEEP_DISRUPTED	**Meaning**: RTM has terminated the task or SRB; no release is necessary. **Action**: None
20 (14)	IEA_SPACE_TERMINATING	**Meaning**: The address space that contains the task or SRB is terminating; no release is necessary. **Action**: None
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. If a current_du_pause_element_token of 16 bytes of binary zeros is specified, one or more locks other than the local lock are held. Otherwise, one or more locks are held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
52 (34)	IEA_ALREADY_SUSPENDED	**Meaning**: The pause element was already paused. **Action**: Check the calling program for a probable coding error and correct the program and rerun it.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
60 (3C)	IEA_AUTH_TOKEN	**Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
68 (44)	IEA_XFER_TO_SELF	**Meaning**: Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
72 (48)	IEA_XFER_FAILED	**Meaning**: The transfer failed, and the current_du_pause_element_token is no longer usable. **Action**: Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.
84 (54)	IEA_INVALID_LINKAGE	**Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 70. IEA4APE — Allocate_Pause_Element

Description

Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	• When supervisor state and PSW key 0: The local lock may be held. • When problem state, or not PSW key 0: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Only 2040 unauthorized PETs may be allocated at any one time in an address space.

Input register information

Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return Code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	

Syntax	Description
CALL IEA4APE	,(return_code ,auth_level ,pause_element_token)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants that are defined in IEAASM or IEAC. The level needed is derived by adding the values of the required types together. The authorization type is required.

For example, the level to allocate authorized pause elements that are checkpoint- or restart-tolerant is IEA_AUTHORIZED + IEA_CHECKPOINTOK, or 3.

The following levels are supported:

Table 71. Authorization

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	When using the allocated pause element through other services, either auth_level IEA_UNAUTHORIZED or IEA_AUTHORIZED can be used.
IEA_AUTHORIZED	1	When using the allocated pause element through other services, auth_level=IEA_AUTHORIZED will be required. Caller must be both key 0 and supervisor state.

Table 72. Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_CHECKPOINTOK	2	The application can tolerate the pause elements' not being restored upon a restart after a checkpoint.

Note: If the IEA_CHECKPOINTOK value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.

,pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that you can use to synchronize the processing of a task.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: **Decimal (Hex)** **Equate symbol**	**Meaning and Action**
00 (0) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
24 (18) IEA_LOCK_HELD	**Meaning:** Program error. If the auth_level indicates AUTHORIZED, locks other than the local lock are held. If the auth_level indicates UNAUTHORIZED, locks are held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
56 (38) IEA_NO_PETS_AVAILABLE	**Meaning**: There are no pause element tokens available. **Action**: Retry the request later.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
68 (44) IEA_XFER_TO_SELF	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
72 (48) IEA_XFER_FAILED	**Meaning**: Environmental error. The system could not obtain storage for a pause element. The system rejects the service call. **Action**: Retry the request later. If the problem persists, consult your system programmer.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 71. IEA4APE2 — Allocate_Pause_Element

Description

Allocate_Pause_Element obtains a pause element token (PET), which uniquely identifies a pause element. The PET is used as input to the following services:

- Pause
- Release
- Transfer
- Deallocate_Pause_Element

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	• For LINKAGE=BRANCH: The local lock may be held. • For LINKAGE=SVC: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Only 2040 unauthorized PETs may be allocated at any one time in an address space.

Allocate_Pause_Element cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEA4APE2	,(return_code ,pause_element_auth_level ,pause_element_token ,pause_element_owner_stoken ,owner_termination_release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Allocate_Pause_Element service.

,pause_element_auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Represents one or more possible levels of the pause element being allocated. The calling program can use the constants defined in IEAASM or IEAC, as appropriate. The level desired results from adding the values of the required types together. The authorization type is not optional.

For instance, the level to allocate authorized pause elements that are checkpoint/restart tolerant is IEA_AUTHORIZED + IEA_CHECKPOINTOK, or 3.

The following levels are supported:

Table 73. Authorization

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	When using the allocated pause element through other services, either pause_element_auth_level IEA_UNAUTHORIZED or IEA_AUTHORIZED can be used.
IEA_AUTHORIZED	1	When using the allocated pause element through other services, pause_element_auth_level =IEA_AUTHORIZED is required. Caller must be both key 0 and supervisor state.

Table 74. Checkpoint/Restart Toleration - only available when the CVTPAUS4 bit is set in the CVT.

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_CHECKPOINTOK	2	The application can tolerate the pause elements' not being restored upon a restart after a checkpoint.

Note: If the IEA_CHECKPOINTOK value is not added to the authorization value, checkpoints cannot be taken when an allocated pause element exists.

,pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies a pause element that you can use to synchronize the processing of a task or SRB.

,pause_element_owner_stoken
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

Specifies the space token (STOKEN) of the address space which is to be considered the owner of the Pause Element being allocated. Specify one of the following values:

- Binary zero: indicate the system should make the current primary address space the owner of the Pause Element. This is the only value valid for key 8-15 problem state callers.
- A valid STOKEN, indicate the system should make the address space with the matching STOKEN the owner for the pause element.

When the CMRO task (the first job step task) of an address space terminates, the system will release and deallocate any pause elements owned by the CMRO task's home address space. The table below describes exactly when the system will release and/or deallocate a Pause Element:

Allocation Service version:	**Deallocation Rules**
IEA4APE	The PE will be deallocated by the system when one of the following events occurs: • The PE was never used to pause a task or SRB and the CMRO task for the space which allocated it terminates. • The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ. • The CMRO task of the home address space of the task or SRB which last used the PE terminates and the PE is not being used to pause an SRB. The home address space of the task or SRB which last used the PE terminates

Allocation Service version:	Deallocation Rules
IEA4APE2	The PE will be deallocated by the system when one of the following events occurs: • The CMRO task of the address space specified by pause_element_owner_stoken terminates. If the PE is being used to pause a DU when the CMRO task terminates, the system will release the DU using the owner_termination_release_code before the PE is deallocated. Note that in this case, the UPET returned will be 16 bytes of binary zeros, an invalid value. • The PE is being used to pause a task or SRB which is asynchronously terminated via CALLRTM TYPE=ABTERM (for example, cancel or detach) or a PURGEDQ • The PE is being used to pause a task or SRB when the home address space of the task or SRB is terminated • The CMRO task of the home address space of the task or SRB which last used the PE terminates and the PE is not being used to pause an SRB The home address space of the task or SRB which last used the PE terminates.Note: A PE is considered as "being used to pause a task or SRB," when the PE is not Reset or Prereleased.

,owner_termination_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Specifies the release code which will be returned to a paused DU if the system deallocates the pause element while it is being used to pause a task or SRB, due to the CMRO task of its owning address space terminating.

Note: If the system deallocates a PE due to its owner terminating while the PE was not being used to pause a task or SRB, future attempts to use the PE will fail with a return code indicating the PETOKEN was stale or the PE is in an invalid state.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Allocate_Pause_Element service routine is to be invoked. The following options are supported:

Table 75. Linkage option		
Variable	**Value (hexadecimal)**	**Meaning**
IEA_LINKAGE_SVC	0	The Allocate_Pause_Element service routine will be invoked via an SVC linkage. This option can be used when in non-cross memory task mode, any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Allocate_Pause_Element service routine will be invoked via a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
00 (0)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. One or more locks other than the local lock are held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
40 (28)	IEA_INVALID_AUTHCODE	**Meaning:** Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
48 (30)	IEA_OUT_OF_STORAGE	**Meaning:** Environmental error. The system could not obtain storage for a pause element. The system rejects the service call. **Action:** Retry the request later. If the problem persists, consult your system programmer.
56 (38)	IEA_NO_PETS_AVAILABLE	**Meaning:** There are no pause element tokens available. **Action:** Try the request again later.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 72. IEA4DPE - Deallocate_Pause_Element

Description

Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	• When supervisor state and PSW key 0: The local lock may be held. • When problem state, or not PSW key 0: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEA4DPE	,(return_code ,auth_level ,pause_element_token)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	This pause element being deallocated must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	This pause element being deallocated must have been allocated with auth_level=IEA_AUTHORIZED.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (00) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04)	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20) IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the specified pause element token specified is invalid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 73. IEA4DPE2 — Deallocate_Pause_Element

Description

Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	• For LINKAGE=BRANCH: The local lock may be held. • For LINKAGE=SVC: No locks may be held.
Control parameters:	Must in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Input register information

Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
SYSSTATE AMODE64=YES	

Syntax	Description
CALL IEA4DPE2	,(return_code ,pause_element_token ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Deallocate_Pause_Element service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Deallocate_Pause_Element service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Deallocate_Pause_Element service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning:** Successful completion **Action:** None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. One or more locks other than the local lock are held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning:** Program error. The pause element associated with the specified pause element token is invalid or has already been paused. A paused PE must be released before it is deallocated. This return code also can indicate that the address space associated with the pause element is ending or has ended and that the system freed the pause element. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for an unauthorized pause element allocated to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 74. IEA4PME2 — 64-bit pause multiple elements service

Description

IEA4PME2 is a callable service that can be used to pause on one or more pause element tokens (PETs). It is entered in 64-bit mode and the addresses in the parameter list are 64 bits long. When the specified number of pause elements (PEs) represented by PETs has been released, you receive control back with the following:

- A list of PETs that you can use to pause on again
- An indication of which PEs were released
- Their release codes.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – If any input PET was allocated as IEA_AUTHORIZED, supervisor state and PSW key 0. – If all input PETs were allocated as IEA_UNAUTHORIZED, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=SASN=HASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEACSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Key 1-15 or problem state callers must specify linkage as IEA_LINKAGE_SVC. IEA_LINKAGE_SVC is limited to pausing upon no more than 1000 PETs.

Pause cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the EXEC PGM=xxx task).

Input register information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Note that register 13 is not required to contain any particular value. See the **workarea** parameter description.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code.

Note that this service saves and restores full 64-bit GPRs.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

There is a maximum number of PETs which can be processed very quickly by IEAVPME2 without doing additional GETMAINs and FREEMAINs. That number is currently 16.

Syntax

Syntax	Description
CALL IEA4PME2	(return_code ,pause_element_token_list ,updated_pause_element_token_list ,release_code_list ,number_of_PETs_in_each_list ,number_of_PEs_to_release ,linkage ,workarea)

Link-edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEA4PME2:

1.
```
LOAD EP=IEA4PME2
Save the 8-byte entry point address with bit 63 changed to 0 (...)
Put the saved entry point address with bit 63 changed to 0 into 64-bit R15
CALL (15),(...)
```

2.
```
LLGT  15,X'10'(0,0)
L     15,X'220'(15,0)
L     15,X'14'(15,0)
L     15,X'104'(15,0)
CALL  (15),(...)
```

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the highest return code from the Pause service (multiple return codes are possible when more than one PET has been specified – see *release_code_list*). When the low-order bit of the return code is on, *release_code_list* contains the return codes for individual PETs rather than release codes.

Note that no pause has actually occurred if the return code is non-zero. In this situation, any PETs which have been released remain released.

,pause_element_token_list
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes times the number of PEs you want to pause on

A list of the PETs identifying the PEs you want to pause on. *Number_of_PETs_in_each_list* specifies how many PETs are in the list.

,updated_pause_element_token_list
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes times the number of PEs you want to pause on

If *return_code* is 0, a list of the PETs returned by Pause Multiple Elements. Each entry corresponds to an entry in the *pause_element_token_list*. For each PE that was released, the system puts an updated PET into this list. For PEs that are not released, the entry contains the PET from the original *pause_element_token_list*.

These new PETs must be used in place of the PETs specified in *pause_element_token_list* or *pause_element_token* on future calls to the Pause, Release, Transfer, or Deallocate_Pause_Element service. The first byte of each entry in *release_code_list* identifies which PEs were released. *Number_of_PETs_in_each_list*specifies how many PETs are in the list.

If *return_code* is not 0, the PETs are not updated and this list is not returned.

If the paused *workunit* was released by the system (the release code is the *owner_termination_release_code* specified on the IEAVAPE2 allocation), the PET returned in that slot will be 16 bytes of binary zeros, an invalid value.

,release_code_list
Returned parameter

- Type: Fullword
- Character Set: N/A
- Length: 4 bytes times the number of PEs you want to pause on

Each entry corresponds to an entry in the *pause_element_token_list*.

If *return_code* is 0, the pause was successful and has been released. For each PE that was Released, the system puts X'01' into the first byte and the release code for that PET into the second through fourth bytes of the full-word entry for that PET. For PEs which are not released, the entry contains binary zeroes.

If *return_code* is 5, 9, D, 21, 35, 3D, or 41, a problem was found with at least one of the PETs specified in *pause_element_token_list*and the pause request did not complete. The individual return code for each PET is in the second through fourth bytes of the *release_code_list* entry for that PET. Each PET with a return code of 0 would have been paused on if all the other PETs in the list had received return codes of 0.If *return_code* contains any other value, the pause request did not complete and *release_code_list* does not contain any meaningful information.

Note: These return codes are the equivalent of return codes 4, 8, C, 20, 34, 3C, and 40 for IEAVPSE2 (Pause single), with the addition of a low-order bit to signify that the *release_code_list*contains individual PET return codes.

,number_of_PETs_in_each_list
Supplied parameter

Number_of_PETs_in_each_list specifies how many entries are in *release_code_list*.

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

This number specifies how many PETs you want to Pause on. This number also specifies the number of entries in *pause_element_token_list*, *updated_pause_element_token_list* , and *release_code_list*. For SVC entry callers, the maximum number that can be specified is 1000.

,number_of_PEs_to_release
Supplied parameter

Number_of_PETs_in_each_list specifies how many entries are in *release_code_list*.

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

This number specifies how many PEs must be released before control is returned back to the issuer of Pause Multiple Elements. This number must be at least 1 and no more than *number_of_PETs_in_each_list*.

Note that if more PEs than this number were released before IEAVPME2 was issued (pre-released), the number of updated PETs in *updated_pause_element_token_list* will be the actual number released.

,linkage

Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Pause service routine is to be invoked. The following options are supported:

Table 76. Linkages

Variable	Value	Meaning
IEA_LINKAGE_SVC	0	The Pause service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Pause service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

,workarea

Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 216 bytes

Specifies a work area on a doubleword boundary in which the Pause service routine will save information about the caller including the caller's registers. This can be the same area that R13 points to if the first 216 bytes of that area can be used as an F7SA savearea.

ABEND codes

None. However, note that you may receive a GETMAIN abend if this service is unable to obtain storage needed to process the PETs.

Return codes

When IEAVPME2 returns control to your program, if GPR 15 contains 0, 5, 9, D, 21, 35, 3D, or 41, *release_code_list* contains information about the status of each PET that was specified. If GPR 15 contains any other value, *release_code_list*does not contain any meaningful information.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
00 (0)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
05 (05)	IEA_PE_TOKEN_BAD	**Meaning:**Program error. The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
09 (09)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
13 (0D)	IEA_DUPLICATE PAUSE	**Meaning:** The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. The caller is holding one or more locks; no locks may be held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
33 (21)	IEA_PE_BAD_STATE	**Meaning:** Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action:** Check the calling program for a probable coding error, such as attempting to perform a pause or transfer using a pause element token that has already been used to pause or transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_ RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
53 (35)	IEA_ALREADY_SUSPENDED	**Meaning:** The pause element was already paused. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
61 (3D)	IEA_AUTH_LEVEL_MISMATCH	**Meaning:** Program error. The caller was in problem state or key 8, but the pause element token was allocated with *pause_element_auth_level*=IEA_AUTHORIZED. The system rejects the service call. **Action:** Program error. The specified pause element token is not valid. The system rejects the service call.
65 (41)	IEA_PE_NOT_HOME_PM	**Meaning:** Program error. The pause element token was for a pause element allocated with *pause_element_auth_level*=IEA_AUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex)	Equate Symbol	Meaning and Action
76 (4C)	IEA_ABENDED_47B	**Meaning:** After an SRB received abend 47B, it invoked IEA4PSE2. It is not valid to invoke IEA4PSE2 after receiving abend 47B. **Action:** Update the calling program to not invoke IEA4PSE2 after abend 47B.
80 (50)	IEA_INSUSPEND_EXIT	**Meaning:** The suspend exit specified on SUSPEND with SPTOKEN of an SRB invoked IEAVPSE. It is not valid to invoke IEA4PSE2 from a suspend exit. **Action:** Update the calling program to not invoke IEA4PSE2 from a suspend exit.
84 (54)	IEA_INVALID_LINKAGE	**Meaning:** Program error. The linkage value specified is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
104 (68)	IEA_INVALID_NUMBER_OF_PETS	**Meaning:** Program error. The number of PETs specified for Pause Multiple was 0 or for SVC entry users more than 1000. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
108 (6C)	IEA_INVALID_NUMBER_TO_ RELEASE	**Meaning:** Program error. The number of PEs to release is 0 or greater than the number of PETs specified for Pause Multiple. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4094 (FFE)	IEA_ERROR_PETS_INVALIDATED	**Meaning:** Pause processing has encountered an error and cannot complete the Pause request. This input PETs have been invalidated. This return code is only issued for Pause Multiple requests. **Action:** Do not return the PETs to the system using the Deallocate Pause Elements services. Note that new PEs must be obtained before attempting to pause again.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Contact IBM support.

Chapter 75. IEA4PSE — Pause service

Description

Call IEA4PSE service to make the current task nondispatchable. After you pause a task, it remains nondispatchable until a release service specifying the same PET is called. That is, the program issuing the pause does not receive control back until after the release occurs.

If a release service specifying the same PET is called before pause, the system returns control immediately to the calling program, and the task is not paused.

When you use pause, it returns an updated PET. Use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program is running auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only pause another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	

Syntax	Description
CALL IEA4PSE	,(return_code ,auth_level ,pause_element_token ,updated_pause_element_token ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Pause service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum level that the specified pause element was allocated with. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	The pause element being paused must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	The pause element being paused must have been allocated with auth_level=IEA_AUTHORIZED.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element being used to pause the current task. You can obtain the PET from the Allocate_Pause_Element service.

When you use a PET in a call to the pause service, you cannot reuse the PET on a second call to pause or on a call to transfer. The pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task; use the new PET the next time when you make a pause request using the same pause element.

,updated_pause_element_token
Returned parameter

- Type: Character string

- Character Set: N/A
- Length: 16 bytes

A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token, which cannot be reused after a successful call to Pause.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task from its paused condition.

ABEND codes

Abend Code	Reason Code	Description
AC7	001A0001	This is an internal error. Contact IBM support.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (00) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04)	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C) IEA_DUPLICATE_PAUSE	**Meaning**: The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20) IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
52 (34) IEA_ALREADY_SUSPENDED	**Meaning**: The pause element was already paused. **Action**: Check the calling program for a probable coding error and correct the program and rerun it.
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 76. IEA4PSE2 — Pause service

Description

Call IEA4PSE2 service to make the current task or SRB nondispatchable. After you pause a task or SRB, it remains nondispatchable until a release or transfer specifying the same PET is called. That is, the program issuing the pause does not receive control back until after the RELEASE or TRANSFER occurs. At that time, the returned release_code contains a value supplied by the associated release or transfer request.

If a release service specifying the same PET is called before pause, the system returns control immediately to the calling program, and the task or SRB is not paused.

When you use pause, it returns an updated PET. Use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Pause cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEA4PSE2	,(return_code ,pause_element_token ,updated_pause_element_token ,release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Pause service.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element being used to pause the current task or SRB. You obtain the PET from the Allocate_Pause_Element service.

When you use a PET in a call to the pause service, you cannot reuse the PET on a second call to pause or on a call to Transfer. The pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to pause the task or SRB; use the new PET the next time when you make a Pause request using the same pause element.

,updated_pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token. This new PET must be used in place of the PET specified in pause_element_token on future calls to the Pause, Release, Transfer, or Deallocate_Pause_Element service.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code is specified by the issuer of the release service, which can release the task or SRB of the paused condition.

,linkage

Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Pause service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Pause service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Pause service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

Abend Code	Reason Code	Description
AC7	001A0001	This is an internal error. Contact IBM support.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None
04 (04)	IEA_PE_TOKEN_BAD	**Meaning:** Program error. The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C)	IEA_DUPLICATE_PAUSE	**Meaning:** The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
32 (20)	IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
52 (34)	IEA_ALREADY_SUSPENDED	**Meaning:** The pause element was already paused. **Action:** Check the calling program for a probable coding error and correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action:** Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
76 (4C)	IEA_ABENDED_47B	**Meaning:** After an SRB received ABEND 47B, it invoked IEA4PSE2. It is not valid to invoke IEA4PSE2 after receiving ABEND 47B. **Action:** Update the calling program to not invoke IEA4PSE2 after ABEND 47B.
80 (50)	IEA_IN_SUSPEND_EXIT	**Meaning:**The suspend exit specified on SUSPEND with SPTOKEN of an SRB invoked IEA4PSE2. It is not valid to invoke IEA4PSE2 from a suspend exit. **Action:** Update the calling program to not invoke IEA4PSE2 from a suspend exit.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 77. IEA4RLS — Release

Description

Call IEA4RLS service to remove a task that has been paused, or to keep a task from being paused. Although a pause element can be used multiple times to pause a task, a pause element token can be used to successfully pause and release a task only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the pause and transfer services.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	May hold the CPU, local or CMS lock.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	

Syntax	Description
CALL IEA4RLS	,(return_code ,auth_level ,target_du_pause_element_token ,target_du_release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return from the Release service.

,auth_level
Supplied Parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level that the specified pause element was allocated with. The calling program can use constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, defined by IEAASM and IEAC. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	The pause element being released must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	The pause element being released must have been allocated with auth_level=IEA_AUTHORIZED.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element used to pause the task. If the PET identifies a pause element that has not been paused, the task is paused. However, the value specified in target_du_release_code is returned to the caller of pause.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of pause or transfer service that used or will use the same PET to pause a task. If your program is not using this code for communication, set this field to zero.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

Return code in: **Decimal (Hex)** **Equate symbol**	**Meaning and Action**
00 (00) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04) IEA_PE_TOKEN_BAD	**Meaning**: The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET be returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10) IEA_SLEEP_DISRUPTED	**Meaning**: RTM has ended the task; no release is necessary. **Action**: None.
20 (14) IEA_SPACE_TERMINATING	**Meaning**: The address space that contains the task is terminating; no release is necessary. **Action**: None.
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20) IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified is not valid or has already been prereleased. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

Chapter 78. IEA4RLS2 — Release

Description

Call IEA4RLS2 service to remove a task or SRB that has been paused, or to keep a task or SRB from being paused.

Although a pause element can be used multiple times to pause a task or SRB, a pause element token can be used to successfully pause and release a task or SRB only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the pause and transfer services.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	• For LINKAGE=BRANCH: The CPU, CMS, or local locks may be held. • For LINKAGE=SVC: No locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Release cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEA4RLS2	,(return_code ,target_du_pause_element_token ,target_du_release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return from the Release service.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element used to pause a task or SRB. If the PET identifies a pause element that has not been paused, the task is paused. However, the value specified in target_du_release_code is returned to the caller of pause.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of pause or transfer service that used or will use the same PET to pause a task or SRB. If the program is not using this code for communication, set this field to zero.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Release service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Release service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Release service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning:** Successful completion. **Action:** None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning:** The specified pause element token is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning:** The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10)	IEA_SLEEP_DISRUPTED	**Meaning:** RTM has terminated the task or SRB; no release is necessary. **Action:** None
20 (14)	IEA_SPACE_TERMINATING	**Meaning:** The address space that contains the task or SRB is terminating; no release is necessary. **Action:** None
24 (18)	IEA_LOCK_HELD	**Meaning:** Program error. The caller is holding one or more locks; other than the local lock, CMS, or CPU lock, no locks may be held. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning:** Program error. The pause element associated with the pause element token specified is invalid or has already been prereleased. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning:** Environmental error. The system release does not support this service. The system rejects the service call. **Action:** Run the program on a system that supports the service.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
44 (2C)	IEA_INVALID_MODE	**Meaning:** Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action:** Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 79. IEA4RPI — Retrieve_Pause_Element_Information service

Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- The authorization level of the pause element
- The address space that currently owns the pause element
- The current state (reset, prereleased, paused, or released) of the pause element
- If the state of the pause element is prereleased or released, the release code of the pause element

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program can do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	• When supervisor state and PSW key 0: Any PASN, any HASN, any SASN • When problem state, or not PSW key 0: PASN=HASN=SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

None.

Input register information

Before calling the Retrieve_Pause_Element_Information service, the caller does not need to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	

Syntax	Description
CALL IEA4RPI	,(return_code ,auth_level ,pause_element_token ,authorization ,owner ,state ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Retrieve_Pause_Element_Information service.

,auth_level
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the caller's authorization level. The following levels are supported: IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which can be used by the calling program.

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The caller is not key 0 and supervisor state.
IEA_AUTHORIZED	1	The caller is both key 0 and supervisor state.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information will be returned. You can obtain the PET from the Allocate_Pause_Element service.

,authorization
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The authorization level of the creator of the pause element specified by the input PET.

One of the following values:

IEAASM and IEAC defined constants	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The caller is not key 0 and supervisor state.
IEA_AUTHORIZED	1	The caller is not key 0 and supervisor state.
IEA_UNAUTHORIZED + IEA_CHECKPOINTOK	2	Unauthorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.
IEA_AUTHORIZED + IEA_CHECKPOINTOK	3	Authorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.

,owner
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

The Stoken of the address space that currently owns the pause element specified by the input PET.

,state
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state might have changed after it was obtained.

State Constant Hexadecimal (Decimal)	Meaning
IEA4_PET_PRERELEASED 1 (1)	The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.
IEA4_PET_RESET 2 (2)	The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.
IEA4_PET_RELEASED 40 (64)	The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transited the PE into the RESET state.
IEA4_PET_PAUSED 80 (128)	A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code is specified by the issuer of the release service, which can release the task or SRB from the paused condition.

Note: The returned value is random if the state parameter is not IEA4_PET_RELEASED or IEA4_PET_PRERELEASED.

ABEND codes

None.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified an unauthorized auth_level type, but a pause element token allocated with an authorized auth_level type was encountered. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
64 (40)	IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified an unauthorized auth_level type, but a pause element token for a pause element allocated to another address space was specified. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 80. IEA4RPI2 — Retrieve_Pause_Element_Information service

Description

Call Retrieve_Pause_Element_Information to get information about a pause element. The information returned includes:

- The authorization level of the pause element
- The address space that currently owns the pause element
- The current state (reset, prereleased, paused, or released) of the pause element
- If the state of the pause element is prereleased or released, the release code of the pause element

An authorized program can use Retrieve_Pause_Element_Information to test the validity of a pause element passed by an unauthorized program. The authorized program can do this to ensure that it does not perform any operation, such as releasing the pause element, unless the unauthorized program is also able to perform the same operation.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Input register information

Before calling the Retrieve_Pause_Element_Information service, the caller does not need to place any information into any register, unless using it in register notation for the parameters, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	

Syntax	Description
CALL IEA4RPI2	,(return_code ,pause_element_token ,linkage ,owner ,current_stoken ,authorization ,state ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Retrieve_Pause_Element_Information service.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information will be returned. You can obtain the PET from the Allocate_Pause_Element service.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The calling program can use the constants defined in IEAASM or IEAC, as appropriate. Add the specified values together to achieve the desired results. For example, to specify linkage branch and untrusted PET, specify IEA_LINKAGE_BRANCH + IEA_UNTRUSTED_PET.

The following options are supported:

Table 77. Linkage variables

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Retrieve_Pause_Element_Information service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and in either problem state or supervisor state.

Table 77. Linkage variables (continued)		
Variable	**Value (hexadecimal)**	**Meaning**
IEA_LINKAGE_BRANCH	1	The Retrieve_Pause_Element_Information service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

Table 78. Untrusted attribute linkage variable		
Variable	**Value (hexadecimal)**	**Meaning**
IEA_UNTRUSTED_PET	2	The Retrieve_Pause_Element_Information service routine is to validate that the input PET is allowed to be used by an unauthorized program. An authorized program should use this if it is validating a PET provided to it by an unauthorized caller.

,owner
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

The Stoken of the address space that currently owns the pause element specified by the input PET.

,current_stoken
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 8 bytes

If the value returned in state is IEA_PET_PAUSED, The stoken of the home address space of the task or SRB which is paused by the specified pause element. If the value in state is not IEA_PET_PAUSED, the information returned in this parameter is undefined.

,authorization
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The authorization level of the creator of the pause element specified by the input PET.

One of the following values:

IEAASM and IEAC defined constants	**Value (hexadecimal)**	**Meaning**
IEA_UNAUTHORIZED	0	The caller is not supervisor state or not key 0.
IEA_AUTHORIZED	1	The caller is supervisor state and key 0.
IEA_UNAUTHORIZED + IEA_CHECKPOINTOK	2	Unauthorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.
IEA_AUTHORIZED + IEA_CHECKPOINTOK	3	Authorized PET that can tolerate the pause elements' not being restored upon a restart after a checkpoint.

,state
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state might have changed after it was obtained.

State Constant Hexadecimal (Decimal)	**Meaning**
IEA4_PET_PRERELEASED 1 (1)	The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.
IEA4_PET_RESET 2 (2)	The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB will be made nondispatchable.
IEA4_PET_RELEASED 40 (64)	The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the Release or Transfer service has released the task or SRB. In either case, control has not been returned to the caller of the Pause or Transfer service. The system has not transited the PE into the RESET state.
IEA4_PET_PAUSED 80 (128)	A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code is specified by the issuer of the release service, which can release the task or SRB from the paused condition.

Note: The returned value is random if the state parameter is not IEA4_PET_RELEASED or IEA4_PET_PRERELEASED.

ABEND codes

None.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28)	IEA_INVALID_AUTHCODE	**Meaning:** Program error. The pause_element_auth_level value specified in the call is not valid. The system rejects the service call. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning:**Program error. The caller was in Problem state or key 8, or specified IEA_UNTRUSTED_PET, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 81. IEA4TPE — Test_Pause_Element service

Description

Call Test_Pause_Element to test a pause element and determine its state. If the state is prereleased or released, the release code of the pause element also is returned.

To ensure minimal overhead when you use the service, Test_Pause_Element establishes no recovery. You are responsible for supplying any needed recovery to handle errors that occur because of the incorrect input pause element tokens or call state errors.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

None.

Input register information

Before calling the Test_Pause_Element service, the caller does not have to place any information into any register, unless using the input register in register notation for the parameters, or using the input register as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEA4TPE	,(return_code ,pause_element_token ,state ,release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Test_Pause_Element service.

,pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element for which information is to be returned. You can obtain the PET from the Allocate_Pause_Element service.

,state
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

The state of the pause element specified by the input PET.

Note: The value returned is the state at the time the service obtained it. The state might have changed after it was obtained.

State Constant Hexadecimal (Decimal)	**Meaning**
IEA4_PET_PRERELEASED 1 (1)	The PE was released before any task or SRB was suspended on it, and no task or SRB has attempted to pause it.
IEA4_PET_RESET 2 (2)	The PE is not being used to make any task or SRB nondispatchable. If the PE is used in an attempt to pause the current task or SRB, the task or SRB is made nondispatchable.
IEA4_PET_RELEASED 40 (64)	The task RB or SRB is currently dispatchable, but control has not been returned to the task or SRB following a call to the Pause or Transfer service. A call to the release or transfer service has released the task or SRB. In either case, control has not been returned to the caller of the pause or transfer service. The system has not change the PE into the RESET state.
IEA4_PET_PAUSED 80 (128)	A task RB or SRB is currently nondispatchable. Its dispatchability is controlled by the PE.

,release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code is specified by the issuer of the Release service, which released the task or SRB from the paused condition.

Note: The returned value is random if the state parameter is not IEA4_PET_RELEASED or IEA4_PET_PRERELEASED.

ABEND codes

None.

Return codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Chapter 82. IEA4XFR — Transfer service

Description

Call IEA4XFR service to release a paused task, and, when possible, give the task immediate control. This service can also, optionally, pause the task under which the transfer request is made. If the caller does not request that its task be paused, the caller's task remains dispatchable.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	• When supervisor state and PSW key 0: Task or SRB. • When problem state, or not PSW key 0: Task.
Cross memory mode:	• For auth_level=IEA_UNAUTHORIZED: PASN=HASN=SASN • For auth_level=IEA_AUTHORIZED: Any PASN, any HASN, any SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	When supervisor state and PSW key 0 and a current_du_pause_element_token of 16 bytes of binary zeros are specified, the local lock may be held. Otherwise, no locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the object code of the calling program with the linkable stub routine (IEA4CSS from SYS1.CSSLIB), or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only transfer to another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input register information

Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	

Syntax	Description
CALL IEA4XFR	,(return_code ,auth_level ,current_du_pause_element_token ,updated_pause_element_token ,current_du_release_code ,target_du_pause_element_token ,target_du_release_code)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the transfer service.

,auth_level
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Indicates the maximum authorization level of the pause element being deallocated. The calling program can use constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, defined by IEAASM and IEAC. The following levels are supported:

Variable	Value (HEX)	Meaning
IEA_UNAUTHORIZED	0	The pause elements must have been allocated with auth_level=_UNAUTHORIZED.
IEA_AUTHORIZED	1	The pause elements must have been allocated with auth_level=_AUTHORIZED.

,current_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element used to pause the current task. When a PET is used on a call to the pause service, it cannot be reused on a second call to pause or as a current_du_pause_element_token on transfer. A new PET is returned to updated_pause_element_token. The new PET now properly defines the pause element and should be used the next time when a pause, transfer, release, or deallocate_pause_element request is using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task will not be paused. The updated_pause_element_token and current_du_release_code are unpredictable.

CAUTION: Do not specify the same PET for both current_du_pause_element_token and target_pause_element_token.

,updated_pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to pause or transfer service.

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

,current_du_release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the release or transfer service that released the current task from the paused condition.

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element to release the target task. Any PET that specifies a pause element not currently being used to pause a task is valid. When a PET for a previously released pause element is used to try to pause a task, the task is not paused; however, the value specified in target_du_release_code will still be returned to the caller of pause or transfer service.

If the task was paused and is now dispatchable, the task will immediately be given control on the current processor.

CAUTION: Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of the pause or transfer service that used (or will use) the same PET to pause a task.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C)	IEA_DUPLICATE_PAUSE	**Meaning**: The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10)	IEA_SLEEP_DISRUPTED	**Meaning**: RTM has terminated the task or SRB; no release is necessary. **Action**: None
20 (14)	IEA_SPACE_TERMINATING	**Meaning**: The address space that contains the task or SRB is terminating; no release is necessary. **Action**: None
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one or more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28)	IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
52 (34)	IEA_ALREADY_SUSPENDED	**Meaning**: The pause element was already paused. **Action**: Check the calling program for a probable coding error and correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
68 (44)	IEA_XFER_TO_SELF	**Meaning:** Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
72 (48)	IEA_XFER_FAILED	**Meaning**: The transfer failed, and the current_du_pause_element_token is no longer useable. **Action**: Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 83. IEA4XFR2 — Transfer service

Description

Call IEA4XFR2 service to release a paused task or SRB, and, when possible, give the task or SRB immediate control. This service can also, optionally, pause the task or SRB under which the transfer request is made. If the caller does not request that its task or SRB be paused, the caller's task or SRB remains dispatchable.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	• For LINKAGE=BRANCH, supervisor state and PSW key 0. • For LINKAGE=SVC: – Working with an IEA_AUTHORIZED pause element, supervisor state and PSW key 0-7. – Working with an IEA_UNAUTHORIZED pause element, problem state and any PSW key.
Dispatchable unit mode:	• For LINKAGE=BRANCH: Task or SRB • For LINKAGE=SVC: Task
Cross memory mode:	• For LINKAGE=BRANCH: Any PASN, any HASN, any SASN • For LINKAGE=SVC: PASN=HASN=SASN
AMODE:	64-bit
RMODE:	Includes 64-bit
ASC mode:	Primary mode.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	If LINKAGE=BRANCH and a current_du_pause_element_token of 16 bytes of binary zeros is specified, the local lock may be held. Otherwise, no locks may be held.
Control parameters:	Must be in the primary address space and addressable by the caller.

Programming requirements

Either link the calling program's object code with the linkable stub routine (IEA4CSS from SYS1.CSSLIB) or load the calling program and then call the service. The high-level language (HLL) definitions for the callable service are:

HLL Definition	Description
IEAASM	390 Assembler declarations
IEAC	C/390 and C++/390 declarations

Restrictions

Pause elements that are created with pause_element_auth_level=IEA_UNAUTHORIZED may only be used by callers in task mode and can only be released from a task in their home address space.

Transfer cannot be used by tasks that are higher in the task tree than the cross memory resource owning task (the top, or first, job step task in the address space).

Input register information

Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of the parameter address list.

13
Address of a 144-byte register save area.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Syntax	Description
SYSSTATE AMODE64=YES	
CALL IEA4XFR2	,(return_code ,current_du_pause_element_token ,updated_pause_element_token ,current_du_release_code ,target_du_pause_element_token ,target_du_release_code ,linkage)

Parameters

The parameters are explained as follows:

return_code
Returned parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Transfer service.

,current_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element that is being or will be used to pause a task or SRB. When a PET is used on a call to the pause service, it cannot be reused on a second call to pause or as a current_du_pause_element_token on transfer. A new PET is returned to update_pause_element_token. The new PET now properly defines the pause element and should be used the next time when a pause, transfer, release, or deallocate_pause_element request is using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task or SRB will not be paused. The updated_pause_element_token and current_du_release_code are unpredictable.

CAUTION: Do not specify the same PET for both current_du_pause_element_token and target_pause_element_token.

,updated_pause_element_token
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to Pause or Transfer.

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

,current_du_release_code
Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the release or transfer service that released the current task or SRB from the paused condition.

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

,target_du_pause_element_token
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies a pause element that is being or will be used to pause a task or SRB. If the task or SRB is paused, it will be released, and, if possible, be given control. If the task or SRB is not paused using the specified pause element, it will not be paused when an attempt to pause is made. In either case the task or SRB will be returned the value specified in target_release_code.

CAUTION: Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

,target_du_release_code
Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of the pause or transfer service used (or will use) the PET specified in target_du_pause_element_token to pause a task or SRB.

linkage
Supplied parameter

- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Specifies how the Transfer service routine is to be invoked. The following options are supported:

Variable	Value (hexadecimal)	Meaning
IEA_LINKAGE_SVC	0	The Transfer service routine will be invoked by an SVC linkage. This option can be used when in non-cross memory task mode, in any key, and either problem state or supervisor state.
IEA_LINKAGE_BRANCH	1	The Transfer service routine will be invoked by a branch instruction. The caller must be in both key 0 and supervisor state. This option must be selected when in SRB mode.

ABEND codes

None.

Return codes

When the service returns control to the resource manager, GPR 15 and the return_code parameter contain a hexadecimal return code.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	IEA_PE_TOKEN_BAD	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08)	IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
12 (0C)	IEA_DUPLICATE_PAUSE	**Meaning**: The work unit has already been paused using the specified pause element token. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10)	IEA_SLEEP_DISRUPTED	**Meaning**: RTM has terminated the task or SRB; no release is necessary. **Action**: None
20 (14)	IEA_SPACE_TERMINATING	**Meaning**: The address space that contains the task or SRB is terminating; no release is necessary. **Action**: None
24 (18)	IEA_LOCK_HELD	**Meaning**: Program error. If a current_du_pause_element_token of 16 bytes of binary zeros is specified, one or more locks other than the local lock are held. Otherwise, one or more locks are held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20)	IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified in the call is not in a valid state. The system rejects the service call. **Action**: Check the calling program for a probable coding error, such as attempting to perform a Pause or Transfer using a pause element token that has already been used to Pause or Transfer by another unit of work. Correct the program and rerun it.
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
52 (34)	IEA_ALREADY_SUSPENDED	**Meaning**: The pause element was already paused. **Action**: Check the calling program for a probable coding error and correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning:** Program error. The caller was in Problem state or key 8, but the pause element token was allocated with pause_element_auth_level=IEA_AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning:** Program error. The pause element token was for a pause element allocated with pause_element_auth_level=IEA_UNAUTHORIZED to another address space. **Action:** Check the calling program for a probable coding error. Correct the program and rerun it.
68 (44)	IEA_XFER_TO_SELF	**Meaning**: Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
72 (48)	IEA_XFER_FAILED	**Meaning**: The transfer failed, and the current_du_pause_element_token is no longer usable. **Action**: Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action:** Search problem reporting databases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Chapter 84. IEECMDS — Query/remove attached commands

Description

This macro provides the same function as the CMDS operator command. It can be used to obtain information about MVS commands which are attached or waiting to be attached in the *MASTER* or CONSOLE address spaces.

It can also be used to remove commands which are waiting. It cannot be used to cancel commands which are already executing.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state. System PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN = SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No Locks may be held.
Control parameters:	Must be in the primary address space.

Programming requirements

The calling program may include mapping macro IEEZB889 which can be used to map the information which is returned in the buffer specified as BUFFER.

Restrictions

The caller cannot be protected by an FRR.

Input register information

Before issuing the IEECMDS macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason Code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14–15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IEECMDS macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEECMDS.
IEECMDS	
ƀ	One or more blanks must follow IEECMDS.
REQUEST=COUNT	**Default:** REQUEST=COUNT
,REQUEST=INFO	
,REQUEST=REMOVE	
,BUFFER=*buffer*	Required with REQUEST=INFO or REQUEST=REMOVE
	buffer: RS-type address or address in register (2) - (12).
,BUFSIZE=*bufsize*	Required with REQUEST=INFO or REQUEST=REMOVE

Syntax	Description
	bufsize: RS-type address or address in register (2) - (12).
,CLASS=*class*	*class*: RS-type address or address in register (2) - (12).
,CLASS=ANY_CLASS	**Default:** CLASS=ANY_CLASS
,CMD=*cmd*	*cmd*: RS-type address or address in register (2) - (12)
,CMD=ANY_CMD	**Default:**CMD=ANY_CMD
,ID =*id*	*id*: RS-type address or address in register (2) - (12)
,ID=ANY_ID	**Default:**: ID=ANY_ID
,JOB=*job*	*job*: RS-type address or address in register (2) - (12)
,JOB=ANY_JOB	**Default:**: CMD=ANY_JOB
,COUNT=*count*	*count*: RS-type address or address in register (2) - (12)
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	*list addr*: RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*NOCHECK)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEECMDS macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,REQUEST=COUNT
,REQUEST=INFO
,REQUEST=REMOVE
An optional parameter that indicates the type of request. The default is REQUEST=COUNT.

- REQUEST=COUNT
  - Return only the count of commands which meet the search criteria.
  - This count is returned for all values of REQUEST, but if REQUEST=COUNT, no other information is returned.
- REQUEST=INFO
  - Return information about commands meeting the search criteria.
  - The count of matching commands is returned.
  - The following information is returned for each command:
    - command names
    - id numbers
    - "waiting or executing" status
    - jobname and asid of the command issuer
    - date/time of issue/execution
- REQUEST=REMOVE
- Remove commands from the "waiting for execution" status, if they meet the search criteria.
- The count of matching commands is returned.
- The following information is returned for each removed command:
  - command names
  - id numbers
  - "waiting or executing" status
  - jobname and asid of the command issuer
  - date/time of issue/execution

Message IEE065I is issued for each removed command. It is directed to the console that issued the removed command.

,BUFFER=*buffer*
A required input parameter if REQUEST=INFO or REQUEST=REMOVE is specified to contain the response.

To code: Specify the RS-type address of address in register (2)-(12), of a character field.

,BUFSIZE=*bufsize*
A required input parameter if REQUEST=INFO or REQUEST=REMOVE is specified to contain the size of the output storage buffer.

To code: Specify the RS-type address of address in register (2)-(12), of a fullword field.

CLASS=*class*
CLASS=ANY_CLASS
An optional input parameter that indicates the class of commands to be processed.

The currently defined classes are:

- Class M1 — commands which are attached to *MASTER*, and may be essential to clearing a backlog of Class M2 commands.
- Class M2 — ordinary attached commands which run in the *MASTER* address space.
- Class M3 — only for SEND commands which run in the *MASTER* address space.
- Class C1 — commands which are attached in CONSOLE, and may be essential to clearing a backlog of Class C2 commands.
- Class C2 — ordinary attached commands which run in the CONSOLE address space.
- Class C3 — only for ROUTE commands which run in the CONSOLE address space.

The default is ANY_CLASS.

For detailed information about command classes, see the description of command flooding in *z/OS MVS System Commands*.

To code: Specify the RS-type address or address in register (2)-(12) of a 4–character field.

CMD=*cmd*
CMD=ANY_CMD
An optional input parameter that indicates the name of command to be processed.

The command name must be specified as the full name, not an abbreviation. This is to conform with the command name returned during the previous execution of the macro with REQUEST=INFO. The default is ANY_CMD.

To code: Specify the RS-type address or address in register (2)-(12) of an 8–character field.

ID=*id*
ID=ANY_ID
An optional input parameter that indicates the id number of the command that had been returned on a previous CMDS INFO command. The default is ANY_ID.

To code: Specify the RS-type address or address in register (2)-(12) of an 4–character field.

JOB=*job*
JOB=ANY_JOB
An optional input parameter that indicates the jobname of the job which issued the commands. The default is ANY_CMD.

To code: Specify the RS-type address or address in register (2)-(12) of an 8–character field.

COUNT=*count*
An required input parameter that contains the address area that will contain the number of commands meeting the specified criteria.

To code: Specify the RS-type address or address in register (2)-(12) of an pointer field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

,list addr
The name of a storage area to contain the parameters. For MF=S MF=E and MF=M, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

IBM recommends that you use the modify and execute forms of IEECMDS in the following order:

1. Use IEECMDS...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
2. Use IEECMDS...MF=(M,list-addr,NOCHECK) specifying the parameters that you want to change.
3. Use IEECMDS...MF=(E,list-addr,NOCHECK) to execute the macro.

ABEND codes

None.

Return codes

Macro IEECMDS provides equate symbols for the return and reason codes.

When the IEECMDS macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
00	—	**Equate Symbol:** CMDS_RC_OK **Meaning**: Matching commands have been found. In the case of a REQUEST type of INFO or REMOVE, the output buffer was sufficient to hold all of the information for the commands meeting the search criteria. **Action**: None
04	—	**Equate Symbol**: CMDS_RC_NOCMDS **Meaning**: No commands meet the specified filters. **Action**: None
08	—	**Equate Symbol:** CMDS_RC_NOSTOR **Meaning**: Insufficient return buffer storage to complete the query operation. **Action**: Refer to the action provided with the specific reason code.
08	04	**Equate Symbol:** CMDS_RS_SOMECMDS **Meaning**: The output buffer is too small to contain all requested information, but does contain the information for one or more commands. If REQUEST=REMOVE, the system has removed only the commands for which information is returned. **Action**: The count of matching commands has been returned. Adjust the buffer size so that it is at least as large as the count multiplied by the output size for each entry, plus the length of the header, and issue the macro again. The constant CMDS_HEADER_LENGTH represents the amount of storage required for the buffer header. The constant CMDS_ENTRY_LENGTH represents the amount of storage required per command. These constants are declared in mapping macro IEEZB889.

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
08	08	**Equate Symbol:** CMDS_RS_NOCMDS **Meaning**: The output buffer is too small to contain the information for even one command. If REQUEST=REMOVE, the system has not removed any commands. **Action**: The count of matching commands has been returned. Adjust the buffer size so that it is at least as large as the count multiplied by the output size for each entry, plus the length of the header, and issue the macro again. The constant CMDS_HEADER_LENGTH represents the amount of storage required for the buffer header. The constant CMDS_ENTRY_LENGTH represents the amount of storage required per command. These constants are declared in mapping macro IEEZB889.
10	—	**Equate Symbol:** CMDS_RC_INVPL **Meaning**: Invalid parameter list. **Action**: Refer to the action provided with the specific reason code
10	04	**Equate Symbol:** CMDS_RS_INVACRN **Meaning**: The acronym in the parameter list was invalid. **Action**: Correct the acronym in the parameter list and issue IEECMDS again.
10	08	**Equate Symbol:** CMDS_RS_INVADDR **Meaning**: An output message is invalid. An ABEND occurred while trying to access storage at an address specified in the parameter list, possibly because that storage is not accessible by the caller, or the storage does not exist. **Action**: Correct the invalid address in the parameter list and issue IEECMDS again.
10	0C	**Equate Symbol:** CMDS_RS_INVBUFFER **Meaning**: The address or length of the buffer in the parameter list was invalid. **Action**: Correct the values of BUFFER or BUFSIZE or both in the parameter list and issue IEECMDS again.
10	10	**Equate Symbol:** CMDS_RS_INVLGTH **Meaning**: The length of the parameter list is invalid. **Action**: Correct the length in the parameter list and issue IEECMDS again.
10	14	**Equate Symbol:** CMDS_RS_INVVERS **Meaning**: The version specified in the parameter list is invalid. **Action**: Correct the version in the parameter list and issue IEECMDS again.
10	18	**Equate Symbol:** CMDS_RS_INVFUNC **Meaning**: The REQUEST type specified in the parameter list is not a valid REQUEST type. **Action**: Correct the REQUEST type in the parameter list and issue IEECMDS again.
10	1C	**Equate Symbol:** CMDS_RS_INVCLASS **Meaning**: The CLASS specified in the parameter list is not a valid CLASS name. **Action**: Correct the CLASS in the parameter list and issue IEECMDS again.

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
10	20	**Equate Symbol:** CMDS_RS_INVID **Meaning**: The ID specified in the parameter list is not a valid value. The ID value must be a decimal number in EBCDIC printable) characters. **Action**: Correct the ID in the parameter list and issue IEECMDS again.
40	—	**Equate Symbol:** CMDS_RS_SYSERR **Meaning**: System Error. This return code is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply it to the appropriate IBM support personnel.
40	04	**Equate Symbol:** CMDS_RS_SYSABEND **Meaning**: An ABEND occurred during processing. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply it to the appropriate IBM support personnel.
40	08	**Equate Symbol:** CMDS_RS_SYSERR **Meaning**: An error occurred during processing. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply it to the appropriate IBM support personnel.

Chapter 85. IEEQEMCS — Query EMCS console

Description

This macro returns information about EMCS consoles in the system.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state. System PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

The calling program may include mapping macro IEEZB887 which can be used to map the information that is returned in the buffer addressed by BUFPTR.

The calling program may also optionally include macro IEEZB888, which contains the declarations for the reason and return codes used by IEEQEMCS.

Restrictions

None.

Input register information

Before issuing the IEEQEMCS macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

If REQUEST=FULL is specified, the service must read the data space of each console being reported on. Depending on the number of consoles, this can degrade performance of the service.

Syntax

The IEEQEMCS macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEEQEMCS.
IEEQEMCS	
ƀ	One or more blanks must follow IEEQEMCS.
REQUEST=COUNT	**Default**: REQUEST=COUNT
REQUEST=SUMMARY	
,BUFPTR=*bufptr*	*bufptr:* RS-type address or address in register (2) - (12).
,BUFSIZE=*bufsize*	*bufsize:* RS-type address or address in register (2) - (12).
,TOKEN=*token*	*token:* RS-type address or address in register (2) - (12).
,RECSIZE=*recsize*	*recsize:* RS-type address or address in register (2) - (12).
REQUEST=INFO	
,BUFPTR=*bufptr*	*bufptr:* RS-type address or address in register (2) - (12).

Syntax	Description
,BUFSIZE=*bufsize*	*bufsize:* RS-type address or address in register (2) - (12).
,TOKEN=*token*	*token:* RS-type address or address in register (2) - (12).
,RECSIZE=*recsize*	*recsize:* RS-type address or address in register (2) - (12).
REQUEST=FULL	
,BUFPTR=*bufptr*	*bufptr:* RS-type address or address in register (2) - (12).
,BUFSIZE=*bufsize*	*bufsize:* RS-type address or address in register (2) - (12).
,TOKEN=*token*	*token:* RS-type address or address in register (2) - (12).
,RECSIZE=*recsize*	*recsize:* RS-type address or address in register (2) - (12).
,STATUS=ACTIVE	**Default**: STATUS=ACTIVE
,STATUS=INACTIVE	
,STATUS=ALL	
,STATUS=BACKLOG	
,STATUS=ERR	
,BKLG_NUM=*bklg_num*	*bklg_num:* RS-type address or address in register (2) - (12).
	Default: BKLG_NUM=10
,CN=*cn*	*cn:* RS-type address or address in register (2) - (12).
,SYS=*sys*	*sys:* RS-type address or address in register (2) - (12).
,KEY=*key*	*key:* RS-type address or address in register (2) - (12).
,AUTH=ANY	**Default**: AUTH=ANY
,AUTH=MASTER	
,AUTH=SYS	
,AUTH=IO	
,AUTH=CONS	
,AUTH=ALL	
,AUTH=INFO	
,AUTH=SYSONLY	
,AUTH=IOONLY	
,AUTH=CONSONLY	
,AUTH=ALLONLY	

Syntax	Description
,AUTH=INFOONLY	
,ATTR=ANY	**Default**: ATTR=ANY
,ATTR=YES	
,ATTR=ROUT	
,ATTR=HC	
,ATTR=AUTO	
,ATTR=AUTON	
,ATTR=MN	
,ATTR=INTIDS	
,ATTR=UNKNIDS	
,ATTR=NONE	
,DOM=ANY	**Default**: DOM=ANY
,DOM=YES	
,DOM=NORMAL	
,DOM=ALL	
,DOM=NONE	
,CONSCNT=*conscnt*	*conscnt:* RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	

Syntax	Description
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEEQEMCS macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=COUNT
REQUEST=SUMMARY
REQUEST=INFO
REQUEST=FULL
An optional parameter that indicates the type of information request. The default is REQUEST=COUNT.

REQUEST=COUNT
Return only the number of EMCS consoles meeting the search criteria.

REQUEST=SUMMARY
Return the number and names of the consoles that meet the search criteria.

,BUFPTR=*bufptr*
An optional input parameter that contains the address of the storage that the console display will be returned in. This field is only valid for a SUMMARY, INFO, or FULL request.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,BUFSIZE=*bufsize*
When BUFPTR=*bufptr* is specified, a required input parameter that contains the size of the storage buffer. This field is only valid when BUFPTR is specified.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,TOKEN=*token*
An optional input parameter that returns the address of an 8-byte token used to return additional EMCS console information on subsequent calls if BUFSIZE is insufficient. This field is only valid for a SUMMARY, INFO, or FULL request.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,RECSIZE=*recsize*
When TOKEN is specified, RECSIZE is a required input parameter that contains the address of a 4-byte output area that will contain the recommended size of the output storage buffer if BUFSIZE is insufficient. This size represents only enough storage to store information about the one console represented by the TOKEN at the time of this call to IEEQEMCS. The RECSIZE contains data only when return code = is IEEQE_RC_NOSTOR.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

REQUEST=INFO
Return the number, names, and console data information, but no message data space statistics for the consoles meeting the search criteria.

REQUEST=FULL
Return the number, names, console data information, and message data space statistics for the consoles meeting the search criteria.

,STATUS=ACTIVE
,STATUS=INACTIVE
,STATUS=ALL
,STATUS=BACKLOG
,STATUS=ERR
An optional parameter that indicates the status of the EMCS consoles to be returned. The default is STATUS=ACTIVE.

,STATUS=ACTIVE
Return only active consoles.

,STATUS=INACTIVE
Return only inactive consoles.

,STATUS=ALL
Return both active and inactive consoles.

,STATUS=BACKLOG
Return consoles with unretrieved messages. The BKLG_NUM keyword can specify the minimum number of unretrieved messages a console must have to be returned.

,STATUS=ERR
Return only EMCS consoles in an error condition.

,BKLG_NUM=*bklg_num*
An optional input parameter that indicates the minimum number of unretrieved messages a console must have for it to be returned on a STATUS(BACKLOG) call.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,CN=*cn*
An optional input parameter that indicates a console name to search for. The name include wildcard characters.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,SYS=*sys*
An optional input parameter that indicates the system name where the console was last activated. The system name may include wildcarded characters.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,KEY=*key*
An optional input parameter that indicates the KEY that was used to activate the console. The key name may include wildcard characters.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,AUTH=ANY
,AUTH=MASTER
,AUTH=SYS
,AUTH=IO
,AUTH=CONS
,AUTH=ALL
,AUTH=INFO
,AUTH=SYSONLY
,AUTH=IOONLY
,AUTH=CONSONLY
,AUTH=ALLONLY
,AUTH=INFOONLY
An optional parameter that indicates console command authority. The default is AUTH=ANY.

,AUTH=ANY
Return consoles with any authority.

,AUTH=MASTER
Return consoles with MASTER authority only.

,AUTH=SYS
Return consoles with SYS authority or MASTER authority.

,AUTH=IO
Return consoles with IO authority or MASTER authority.

,AUTH=CONS
Return consoles with CONS authority or MASTER authority.

,AUTH=ALL
Return consoles with IO, SYS, and CONS authority, or MASTER authority.

,AUTH=INFO
Return consoles with INFO, IO, SYS, CONS, or MASTER authority.

,AUTH=SYSONLY
Return consoles with SYS authority only.

,AUTH=IOONLY
Return consoles with IO authority only.

,AUTH=CONSONLY
Return consoles with CONS authority only.

,AUTH=ALLONLY
Return consoles with IO, SYS, and CONS authority only.

,AUTH=INFOONLY
Return consoles with INFO authority only.

,ATTR=ANY
,ATTR=YES
,ATTR=ROUT
,ATTR=HC
,ATTR=AUTO
,ATTR=AUTON
,ATTR=MN
,ATTR=NONE
,ATTR=INTIDS
,ATTR=UNKNIDS
An optional parameter that indicates routing attributes of the console. The default is ATTR=ANY.

,ATTR=ANY
Return consoles regardless of routing attributes.

,ATTR=YES
Return consoles that are receiving some type of undelivered messages.

,ATTR=ROUT
Return consoles receiving any routing codes.

,ATTR=HC
Return consoles receiving the hardcopy message set.

,ATTR=AUTO
Return consoles receiving AUTO(YES) messages.

,ATTR=AUTON
Return consoles not receiving AUTO(YES) messages.

,ATTR=MN
Return consoles receiving any type of MONITOR messages.

,ATTR=NONE
Return consoles with no routing attributes.

,ATTR=INTIDS
Return consoles receiving messages directed to console id zero.

,ATTR=UNKNIDS
Return consoles receiving messages directed to "unknown" console ids, such as consoles with one-byte id.

,DOM=ANY
,DOM=YES
,DOM=NORMAL
,DOM=ALL
,DOM=NONE
An optional parameter that indicates the DOM attribute of the consoles. The default is DOM=ANY.

,DOM=ANY
Return consoles regardless of their DOM attribute.

,DOM=YES
Return consoles that are receiving DOMs (either DOM(NORMAL) or DOM(NONE) consoles).

,DOM=NORMAL
Return consoles that are DOM(NORMAL) only.

,DOM=ALL
Return consoles that are DOM(ALL) only.

,DOM=NONE
Return consoles that are DOM(NONE) only.

,CONSCNT=*conscnt*
An optional input parameter that contains the address of a 4-byte output area that will contain the number of consoles meeting the specified criteria. The output area is only filled in for a COUNT request.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,RETCODE=*retcode*
An optional parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the

macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
A required input parameter that specifies the macro form.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of IEEQEMCS in the following order:

- Use IEEQEMCS ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use IEEQEMCS ...MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
- Use IEEQEMCS ...MF=(E,list-addr,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters. For MF=E and MF=M, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IEEQEMCS macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

Table 79. Return and Reason Codes for the IEEQEMCS Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: IEEQE_RC_OK **Meaning**: EMCS consoles have been found. In the case of a REQUEST type of SUMMARY, INFO, or FULL, the output buffer was large enough sufficient to hold all of the information for the consoles meeting the search criteria. **Action**: None required.
4	—	**Equate Symbol**: IEEQE_RC_NOCONS **Meaning**: No EMCS consoles meet the specified filters. **Action**: None required.
8	—	**Equate Symbol**: IEEQE_RC_NOSTOR **Meaning**: Insufficient return buffer storage to complete the query operation. **Action**: Refer to the action provided with the specific reason code.
8	4	**Equate Symbol**: IEEQE_RS_TOKSZCONS **Meaning**: A token and recommended buffer size have been returned in TOKEN and RECSIZE. Also, some console information has been returned in the output buffer. **Action**: Process the information returned in the console buffer, then issue IEEQEMCS again with the token that was returned by this call to IEEQEMCS to obtain more console information.

Table 79. Return and Reason Codes for the IEEQEMCS Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	8	**Equate Symbol**: IEEQE_RS_TOKSZNOCONS **Meaning**: A token and recommended buffer size have been returned in TOKEN and RECSIZE. The output buffer is too small to return any EMCS console information. **Action**: Allocate a new buffer that is at least the size returned in RECSIZE, and issue IEEQEMCS again with the new buffer and the token returned on the previous IEEQEMCS call. The recommended buffer size returned in RECSIZE is sufficient to hold only one console. It may be necessary to obtain a buffer larger than that to hold all of the consoles returned by IEEQEMCS.
8	12	**Equate Symbol**: IEEQE_RS_NOTOKSZRET **Meaning**: TOKEN and RECSIZE parameters were not coded on the macro invocation, so IEEQEMCS could not return a recommended buffer size to the caller. The buffer size specified by BUFSIZE was not sufficient to hold all of the consoles returned by IEEQEMCS. **Action**: Issue IEEQEMCS again with the TOKEN and RECSIZE parameters.
12	—	**Equate Symbol**: IEEQE_RC_INVTOK **Meaning**: Invalid token in parameter list. **Action**: Issue IEEQEMCS again with a correct token or a token of zeros.
16	—	**Equate Symbol**: IEEQE_RC_INVPL **Meaning**: Invalid parameter list. **Action**: Refer to the action provided with the specific reason code.
16	4	**Equate Symbol**: IEEQE_RS_INVACRN **Meaning**: The eyecatcher (ECDM) in the parameter list was invalid. **Action**: Correct the eyecatcher (ECDM) in the parameter list and issue IEEQEMCS again.
16	8	**Equate Symbol**: IEEQE_RS_INVADDR **Meaning**: An output address is invalid. An ABEND occurred while trying to access storage at an address specified in the parameter list, possibly because that storage is not accessible by the caller, or the storage does not exist. **Action**: Correct the address in the parameter list and issue IEEQEMCS again.
16	12	**Equate Symbol**: IEEQE_RS_INVBUFSIZEADDR **Meaning**: The BUFSIZE parameter was invalid. **Action**: Correct the BUFSIZE parameter and issue IEEQEMCS again.
16	16	**Equate Symbol**: IEEQE_RS_INVLGTH **Meaning**: The length of the parameter list is invalid. **Action**: Correct the length in the parameter list and issue IEEQEMCS again.
16	20	**Equate Symbol**: IEEQE_RS_INVVERS **Meaning**: The version specified in PLISTVER is invalid. **Action**: Correct the version and issue IEEQEMCS again.
16	24	**Equate Symbol**: IEEQE_RS_INVFUNC **Meaning**: The REQUEST type specified in the parameter list is not a valid REQUEST type. **Action**: Correct the REQUEST type in the parameter list and issue IEEQEMCS again.

Table 79. Return and Reason Codes for the IEEQEMCS Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
16	28	**Equate Symbol**: IEEQE_RS_INVSTAT **Meaning**: The STATUS type specified in the parameter list is not a valid STATUS type. **Action**: Correct the STATUS type in the parameter list and issue IEEQEMCS again.
16	32	**Equate Symbol**: IEEQE_RS_INVAUTH **Meaning**: The command authority specified in the parameter list is not a valid command authority type. **Action**: Correct the AUTH value in the parameter list and issue IEEQEMCS again.
16	36	**Equate Symbol**: IEEQE_RS_INVDOM **Meaning**: The DOM attribute specified in the parameter list is not a valid DOM attribute type. **Action**: Correct the DOM attribute in the parameter list and issue IEEQEMCS again.
16	40	**Equate Symbol**: IEEQE_RS_INCONSIST **Meaning**: A set of parameters specified in the parameter list conflict with each other. **Action**: Correct the parameter list to avoid conflicting parameters and issue IEEQEMCS again.
16	44	**Equate Symbol**: IEEQE_RS_INVATTR **Meaning**: The routing attributes specified in the parameter list are not valid routing attribute types. **Action**: Correct the ATTR field in the parameter list and issue IEEQEMCS again.
64	—	**Equate Symbol**: IEEQE_RC_SYSERR **Meaning**: System Error. This return code is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.
64	04	**Equate Symbol**: IEEQE_RS_SYSABEND **Meaning**: An ABEND occurred during processing. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.
64	08	**Equate Symbol**: IEEQE_RS_SYSERR **Meaning**: An error occurred during processing. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply it to the appropriate IBM support personnel.

Examples

Example

Operation: This example requests FULL information about all consoles on system SYS01 that are in an error condition.

```
         .
         .
*        Initialize BUFSIZE and TOKEN
         L     REG9,INITSIZE
```

```
          ST    REG9,BUFSIZE
          MVC   TOKEN,INITTOKEN
          Set up addresses for IEEQEMCS
          LA    REG9,TOKEN
          ST    REG9,TOKENPTR
          LA    REG9,RECSZ
          ST    REG9,RECSZPTR
*         Get storage for output buffer
          STORAGE OBTAIN,LENGTH=BUFSIZE,ADDR=BUFPTR,COND=NO,LOC=ANY
*         Issue IEEQEMCS
DOQEMCS   EQU   *
          IEEQEMCS REQUEST=FULL,        Full info                     X
               STATUS=ALL,              Active or inactive consoles   X
               TOKEN=TOKENPTR,          Token                         X
               BUFPTR=BUFPTR,           Buffer address                X
               BUFSIZE=BUFSIZE,         Buffer size                   X
               RECSIZE=RECSZPTR,        Recommended size              X
               RETCODE=RETCODE,         Return code                   X
               RSNCODE=RSNCODE,         Reason code                   X
               MF=(E,PLIST,COMPLETE)
*         Check return and reason codes from IEEQEMCS
          LA  REG14,IEEQE_RC_OK       Check if all consoles have been
          C   REG14,RETCODE             returned
          BE  ALLOK                   All consoles have been returned,
                                        so process them
          LA  REG14,IEEQE_RC_NOCONS   Check if no consoles have been
          C   REG14,RETCODE             returned
          BE  DONE                    No consoles met the filter, so   X
                                        exit
          LA  REG14,IEEQE_RC_NOSTOR   Check if the storage buffer could
          C   REG14,RETCODE             not hold all the consoles
          BNE ERROR                   No, there was some kind of error
          LA  REG14,IEEQE_RS_TOKSZCONS
          C   REG14,RSNCODE
          BE  SOMEOK                  The buffer was too small to hold X
                                      all the consoles meeting this    X
                                      filter, but IEEQEMCS put as many X
                                      consoles as possible in the      X
                                      buffer.  Process the consoles,   X
                                      and call IEEQEMCS again.
          LA  REG14,IEEQE_RS_TOKSZNOCONS
          C   REG14,RSNCODE
          BE  GETMORE                 The buffer was too small for     X
                                      even one console.  Free the      X
                                      buffer, get more storage, and    X
                                      call IEEQEMCS again.
          .
          .
          .
ALLOK     EQU *
*         Call PROCCONS to process the output buffer.
          LA   REG15,PROCCONS
          BALR REG14,REG15
*         There are no more consoles to process, so exit
          XR   REG15,REG15            Zero return code
          B    DONE
SOMEOK    EQU *
*         Call PROCCONS to process the consoles returned in
*         the output buffer.
          LA   REG15,PROCCONS
          BALR REG14,REG15
*         There is more console information to be retrieved, so
*         loop back to DOQEMCS.
          B    DOQEMCS
PROCCONS  EQU  *
*         Process consoles, and return to caller.
          ST   REG14,RETADDR
          L    HDRREG,BUFPTR          Load pointer to buffer in R2
          USING ECDM_HDR,HDRREG       This should be the ECDM header   X
                                      pointer
          L    SUMMREG,ECDM_HDR_SIZE  Load the size of the header
          AR   SUMMREG,HDRREG         Find the address of the summary  X
                                      block
          LH   REG15,ECDM_NENT        Load number of entries
          ST   REG15,NUMCONS
          USING ECDM_SUMM,SUMMREG
          USING ECDM_INFO,INFOREG
          USING ECDM_DSP,DSPREG
          USING ECDM_CNSW,CNSWREG
          USING ECDM_MSCP,MSCPREG
CONSLOOP  LTR  REG15,REG15            Is the number of entries zero?
          BZ   CONSDONE               Yes, exit loop
```

```
          L    REG8,ECDM_SUMM_SIZE    Put size of summ block in R10
          L    ENDREG,ECDM_SUMM_CONS_SIZE Put size of all blocks for   X
                                          this console in R11
          AR   REG8,SUMMREG           Get address of end of the        X
                                      summary block in R8 (work reg)
          AR   ENDREG,SUMMREG         Get address of the end of the    X
                                      console block in R11
BLKLOOP   LR   REG9,REG8              Copy work addr into R9
          SR   REG9,ENDREG            Are we at the end of the block?
          BZ   DOCONS                 Yes, do the actual console       X
                                      processing
          LA   REG9,ECDM_TYPE_INFO    Load the type of block in R9
          CH   REG9,0(REG8)           Is this an info block?
          BE   INFOBLK                Yes, branch to info block
          LA   REG9,ECDM_TYPE_DSP     Load the type of block in R9
          CH   REG9,0(REG8)           Is this a DSP block?
          BE   DSPBLK                 Yes, branch to data space block
          LA   REG9,ECDM_TYPE_MSCP    Load the type of block in R9
          CH   REG9,0(REG8)           Is this an MSCOPE block?
          BE   MSCPBLK                Yes, branch to MSCOPE block
          B    ERROR                  There was a bad type, so exit
INFOBLK   LR   INFOREG,REG8           Load INFO block addr into R4
          A    REG8,ECDM_INFO_SIZE    Increment size of block
          B    BLKLOOP                Find next block
DSPBLK    LR   DSPREG,REG8            Load DSP block addr into R5
          A    REG8,ECDM_DSP_SIZE     Increment size of block
          B    BLKLOOP                Find next block
MSCPBLK   LR   MSCPREG,REG8           Load MSCP block addr into R6
          A    REG8,ECDM_MSCP_SIZE    Increment size of block
          B    BLKLOOP                Find next block
DOCONS    EQU  *                      Console processing
*         At this point, do any processing on this individual
*         console that is necessary.
          .
          .
          .
*         Now, decrement the console count, and process the
*         next console in the buffer (if there is one).
          L    REG15,NUMCONS
          BCTR REG15,0                Decrement console count
          ST   REG15,NUMCONS
          LR   SUMMREG,REG8           Get address of next block in R3
          B    CONSLOOP               process next block
CONSDONE  L    REG14,RETADDR
          BR   REG14
          .
          .
          .
GETMORE   EQU  *
*         There was not enough storage to complete the request, so
*         get some more.
*         First delete the old storage
          STORAGE RELEASE,LENGTH=BUFSIZE,ADDR=BUFPTR
*         Now, since IEEQEMCS returned a recommended storage
*         size, move that amount into BUFSIZE, and obtain the
*         storage buffer.
          MVC   BUFSIZE,RECSZ
          STORAGE OBTAIN,LENGTH=BUFSIZE,ADDR=BUFPTR,COND=NO,LOC=ANY
*         Loop to issue IEEQEMCS again
          B     DOQEMCS
          .
          .
          .
ERROR     EQU   *
*         Do any error handling here
          .
          .
          .
DONE      EQU   *
*         IEEQEMCS found no more consoles, so release the output
*         buffer.
          STORAGE RELEASE,LENGTH=BUFSIZE,ADDR=BUFPTR
*         Now, continue on with other processing, etc.
          .
          .
          .
*         Declare constants and variables
BUFSIZE   DS    F
BUFPTR    DS    F
TOKENPTR  DS    F
RECSZPTR  DS    F
TOKEN     DS    CL8
```

```
RECSZ     DS    F
RETCODE   DS    F
RSNCODE   DS    F
NUMCONS   DS    F
RETADDR   DS    F
INITSIZE  DC    XL4'300'
INITTOKEN DC    XL8'0'
*         IEEQEMCS parameter list
          DS    0F
          IEEQEMCS MF=(L,PLIST)
*         IEEQEMCS return code equates
          IEEZB888
*         Include mapping of ECDM blocks
          IEEZB887
          END
```

Chapter 86. IEEVARYD — Vary one or more devices online or offline

Description

IEEVARYD varies one or more devices online or offline on a single system, or defines the automatically switchable attribute for a device that supports automatic tape switching. It has the same effect as the VARY device or VARY AUTOSWITCH operator command, but it provides return and reason codes to the calling program, rather than issuing messages to a console.

See *z/OS HCD Planning* for more information about automatic tape switching, and *z/OS Planning for Installation* for the devices that support automatic tape switching.

Comparison to MGCRE macro

The MGCRE macro also allows a program to issue the VARY command. (MGCRE allows a program to issue *any* command.) However, MGCRE automatically checks the SAF authority of the calling program. Also, MGCRE returns text responses to a console specified by the calling program, whereas IEEVARYD provides a return and reason code.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM keys 0-7
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

- The caller must be in a non-swappable address space.
- The IEEVARYD macro does no authorization checking (SAF is not invoked). If you require that the calling program's authority to perform a vary device operation be checked, issue RACROUTE REQUEST=AUTH before issuing the IEEVARYD macro.
- The calling program must perform any logging of information associated with the operation, such as an entry in the system log (SYSLOG).
- You must include the IEEZB833 mapping macro and, if you specify the optional RESULTS keyword, the IEEZB834 mapping macro.
- You can change the automatically switchable characteristic of a tape device only if the device is offline.
- The VDEV_ENQS_HELD flag that allowed the calling program to hold the SYSIEFSD.VARYDEV and SYSIEFSD.Q4 resources is no longer supported. The ENQs must be released before invoking IEEVARYD or an abend 077- 003C will result.

Restrictions

- The VDEV_ENQS_HELD flag that allowed the calling program to hold the SYSIEFSD.VARYDEV and SYSIEFSD.Q4 resources is no longer supported.
- The ENQs must be released before invoking IEEVARYD or an abend 077- 003C will result.

Input register information

Before issuing the IEEVARYD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IEEVARYD macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.

Syntax	Description
ƀ	One or more blanks must precede IEEVARYD.
IEEVARYD	
ƀ	One or more blanks must follow IEEVARYD.
OPERATION=*operation parm*	*operation parm*: RS-type address or register (2) - (12).
,DEVICES=*devices parm*	*devices parm*: RS-type address or register (2) - (12).
,NUMDEVS=*num of devices*	*num of devices*: RS-type address or register (2) - (12).
,RESULTS=*vary results*	*vary results*: RS-type address or register (2) - (12).
,RETCODE=*return code*	*return code*: RS-type address or register (2) - (12).
,RSNCODE=*reason code*	*reason code*: RS-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

OPERATION=*operation parm*
Specifies the address or name of a required 16-byte input area in which you place the VARY device service portion of the initialized VDEV data area, mapped by the IEEZB833 mapping macro. The VARY device service portion is the VDEV control block.

,DEVICES=*devices parm*
Specifies the address or name of a required variable-length area in which you place the IEEVARYD device array entries. Each device entry is mapped by the VDEVARR DSECT of the IEEZB833 mapping macro.

,NUMDEVS=*num of devices*
Specifies the name or address of an optional fullword input area where you specify the number of devices in the device array specified on the DEVICES parameter.

The default is 1.

,RESULTS=*vary results*
Specifies the name or address of an optional array (mapped by the IEEZB834 macro) that contains results for the VARY operation on each device specified in the DEVICES parameter.

,CALLERID=*caller id*
Specifies the name or address of an optional 8-character input area into which you place the name of the caller associated with the VARY operation. If CALLERID is specified, then MVS inserts the identifier into the one or more messages that MVS issues to hardcopy, which indicate that a device was brought

online or taken offline. For example, if a program uses IEEVARYD to VARY device 205 online and specifies XYZ as the CALLERID, MVS issues the following message to hardcopy:

```
IEE302I 0205    ONLINE BY XYZ
```

If the program did not specify the caller ID, the message to hardcopy would be:

```
IEE302I 0205    ONLINE
```

,RETCODE=*return code*
Specifies an optional fullword output area into which IEEVARYD will copy a return code from GPR 15.

,RSNCODE=*reason code*
Specifies an optional fullword output area into which IEEVARYD will copy a reason code from GPR 0.

ABEND codes

The IEEVARYD macro abnormally terminates with abend code 077- 003C if the ENQs are not released before invoking IEEVARYD.

Return and reason codes

When IEEVARYD returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code. If you specified the RETCODE or RSNCODE parameters, those areas will also contain a return code and reason code, respectively.

The following table identifies return code and reason code combinations, tells what each means, and recommends an action that you need to take.

Table 80. Return and Reason Codes for the IEEVARYD Macro

Return Code	Reason Code	Meaning and Action
00	00	**Meaning**: Processing completed successfully. **Action**: No action needed.
04	None.	**Meaning**: The operation was performed against all specified devices, but MVS set the VDRSARR_RETCODE field in the IEEZB834 mapping macro to a non-zero value for at least one device in the device array. **Action**: In the RESULTS area, check the VDRSARR_RETCODE values for each device to determine the result of the operation on each device.
08	01	**Meaning**: Program error. MVS could not access the caller's parameter list. **Action**: Ensure that you have specified the correct parameter list area on the execute form of the macro.
08	02	**Meaning**: Program error. MVS could not access the storage area specified on the DEVICES parameter. **Action**: Ensure that the DEVICES parameter correctly specifies the name or address of storage that contains the device array. The entire device array must reside in storage that is accessible to the program invoking the IEEVARYD macro.
08	03	**Meaning**: Program error. MVS could not access the storage area specified on the OPERATION parameter. **Action**: Ensure that the OPERATION parameter correctly specifies the name or address of storage that contains the VDEV control block. The VDEV control block must reside in storage that is accessible to the program invoking the IEEVARYD macro.

Table 80. Return and Reason Codes for the IEEVARYD Macro (continued)

Return Code	Reason Code	Meaning and Action
08	04	**Meaning**: Program error. MVS could not access the storage area specified on the RESULTS parameter. **Action**: Ensure that the RESULTS parameter correctly specifies the name or address of storage that contains the device results array. The entire device results array must reside in storage that can be updated by the program invoking the IEEVARYD macro.
08	05	**Meaning**: Program error. The input parameter list includes an invalid combination of VARY command keywords. For example: • Both ONLINE and OFFLINE • Neither ONLINE nor OFFLINE • Both OFFLINE and RESET • AUTOSWITCH,ON or AUTOSWITCH,OFF with any other parameters. **Action**: Ensure that the OPERATION parameter identifies a VDEV data area that specifies flags in the VDEV control block that are valid according to guidelines of the VARY command. For information about the VARY command, see *z/OS MVS System Commands*.
08	06	**Meaning**: Program error. The input parameter list includes an invalid combination of VARY COMMAND options. For example: • KEEP_OFFLINE with ONLINE **Action**: Ensure that the OPERATION parameter identifies a VDEV data area that specifies flags in the VDEV control block that are valid according to guidelines of the VARY command. For information about the VARY command, see *z/OS MVS System Commands*.
08	07	**Meaning**: Program error. The NUMDEVS value is not valid. **Action**: Change NUMDEVS to a valid value. Valid values are 1 to 65536.
0C	00	MVS was temporarily unable to process the requested operation. The caller requested the VDEV_DO_NOT_WAIT_FOR_ENQ option, and the IEEVARYD service was unable to obtain the SYSIEFSD.Q4 resource in a reasonable amount of time. The operation might be successful if retried at a later time.
0C	04	MVS was temporarily unable to process the requested operation. The caller requested the VDEV_DO_NOT_WAIT_FOR_ENQ option, and the IEEVARYD service was unable to obtain the SYSIEFSD.VARYDEV resource in a reasonable amount of time. The operation might be successful if retried at a later time.
10	None.	**Meaning**: System error. Some devices were processed, and some were not processed. **Action**: Check the RESULTS area to determine which devices were processed. Record this code and supply it to IBM support personnel.

Examples

Example 1 shows how you define tape devices 200 and 300, both in a varied-offline state, as automatically switchable. The second example then varies the devices online.

Example 1

Use the IEEVARYD macro to define devices 200 and 300 as automatically switchable. The example includes steps to:

1. Initialize the IEEVARYD input for the operation by setting the appropriate keyword and option flags.
2. Initialize the IEEVARYD device array entries with the device number of each device upon which the operation is to be performed.

3. Issue the execute form of the IEEVARYD, specifying the VARY device service input, IEEVARYD device array, VARY device service results, and the IEEVARYD parameter list which was defined when the list form of IEEVARYD was issued.
4. Examine the return code returned in register 15 to determine the overall result of the operation.
5. Use the IEEZB834 mapping macro to determine the results of the operation for each device in the IEEVARYD device array.
6. Free the storage for the IEEVARYD input, IEEVARYD device array, and VARY device service results.

```
*********************************************************************
*
*  Issue the list form of the IEEVARYD to define the IEEVARYD macro
*  parameter list.
*
*  Include the IEEZB833 and IEEZB834 mapping macros in the program
*  declarations.
*
*  IEEZB833 maps the IEEVARYD input including the IEEVARYD device array
*  (VDEVARR).  IEEZB834 maps the VARY device service results.
*
*  Obtain storage for the IEEVARYD input, and the VARY device service
*  results.  Obtain storage for an IEEVARYD device array for each
*  device affected by a single invocation of IEEVARYD.
*  (This example uses two IEEVARYD device arrays.)
*  The address of the storage area is in R1.
*
*  Initialize the IEEVARYD input:
*
         LR    R2,R1              Address of storage
         LA    R4,0               Set to zero for MVCL
         LA    R5,0               Set to zero for MVCL
         MVCL  R2,R4              Clear storage
         USING VDEV,R2            Obtain addressability
         L     R0,CBID            Load identifier
         ST    R0,VDEV_ID         Initialize the identifier
         MVI   VDEV_VERSION,VDEV_VERN Initialize the version number
         OI    VDEV_KEYWORDS1,VDEV_AUTOSWITCH Initialize the operation
*                                 to AUTOSWITCH
         OI    VDEV_KEYWORDS2,VDEV_ON Indicate to turn it ON
         LA    R0,VDEV_LENGTH     Length of the IEEVARYD input
         LR    R4,R2              Address of the IEEVARYD input
         ALR   R4,R0              Address of storage immediately
*                                 following the IEEVARYD input
         LR    R5,R4              Save address of the
*                                 IEEVARYD device array
*
*  Initialize the first IEEVARYD device array entry
*
         USING VDEVARR,R4         Obtain addressability
         LA    R0,200
         STH   R0,VDEVARR_DEVN    Initialize the device number (200)
*
*  Initialize the second IEEVARYD device array entry
*
         LA    R0,VDEVARR_LENGTH  Length of IEEVARYD device array entry
         ALR   R4,R0              Obtain addressability to next entry
         LA    R0,300
         STH   R0,VDEVARR_DEVN    Initialize the device number (300)
*                                 following the IEEVARYD input
*********************************************************************
*
*  Get address of IEEVARYD Results
*
*********************************************************************
         ALR   R4,R0              Address of VDRSARR
*********************************************************************
*
*  Issue the execute form of IEEVARYD to vary the devices online
*
*********************************************************************
         IEEVARYD OPERATION=(R2),DEVICES=(R5),NUMDEVS=#DEVS,           X
               RESULTS=(R4),CALLERID=VDEVICES_ID,MF=(E,IEEVARYL)
*
*  Determine if the operation was not performed because of an error in
*  the parameters (Register 15 = 8)
*
         LA    R0,8
         CR    R15,R0
```

```
         BE    FREEVDEV           If the parameters are in error, free
*                                 the storage for the vary device array
*                                 header and vary device array entries
         USING VDRSARR,R4         Obtain addressability to RESULTS
         LA    R7,1               Initialize counter for loop
*
*  Perform the following loop for each IEEVARYD device array entry to
*  determine the results of the operation against each device
*
LOOP     DS    0H
         C     R7,#DEVS          All entries processed?
         BH    ENDLOOP
         TM    VDRSARR_OUTPUT_FLAGS1,VDRSARR_OUTPUT_VALID Determine if
*                                 output was returned for the device
         BZ    ITERATE            No output for the device, so iterate
         L     R9,VDRSARR_RETCODE Get return code for the device
         LA    R0,VDRSARR_ALREADY_OK Set register 0 to
*                                 highest good return code
         CR    R9,R0              Determine if the operation was
*                                 successful against the device
         BH    BADRETC
GOODRETC DS    0H                 The operation was successful against
*                                 the device
         B     CHECKMSG
BADRETC  DS    0H                 The operation was not successful
*                                 against the device
CHECKMSG DS    0H                 Determine if a message was returned
*                                 for the device
         TM    VDRSARR_OUTPUT_FLAGS1,VDRSARR_MSG_RETURNED
         BZ    NOMSG
MSG      DS    0H                 A message was returned for the device
         B     ITERATE
NOMSG    DS    0H                 A message was not returned for the
*                                 device
ITERATE  DS    0H                 Prepare for the next iteration
         LA    R0,1
         ALR   R7,R0              Increment loop counter
         LA    R0,VDRSARR_LENGTH  Length of IEEVARYD device array entry
         ALR   R4,R0              Obtain addressability to next entry
         B     LOOP               Iterate
ENDLOOP  DS    0H                 End of loop
*
*  Release the storage for the IEEVARYD input, two
*  IEEVARYD device array entries, and IEEVARYD Results
*
FREEVDEV DS    0H
*********************************************************************
*                                                                   *
*  Declarations                                                     *
*                                                                   *
*********************************************************************
CBID     DC    C'VDEV'            Control block identifier for the
*                                 IEEVARYD input
VDEVICES_ID DC C'VDEVICES'        Caller identifier for the IEEVARYD
*                                 Input
@DATA    DS    0H
@DATD    DSECT
         DS    0F
#DEVS    DS    F
*********************************************************************
*
*  Issue the list form of IEEVARYD to define the parameter list
*
*********************************************************************
         IEEVARYD MF=(L,IEEVARYL)
@ENDDATD DS    0X
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
```

```
*********************************************************************
*
*  Include IEEZB833 to define the IEEVARYD input (VDEV) and
*  IEEVARYD device array entry (VDEVARR). Include IEEZB834
*  to define the IEEVARYD Results (VDRSARR).
*
*********************************************************************
         IEEZB833
         IEEZB834
         END   VDEVICES
```

Example 2

The following example illustrates how a program can use IEEVARYD to vary devices 200 and 300 online. The example includes steps to:

1. Issue the list form of IEEVARYD to define the IEEVARYD macro parameter list.
2. Include the IEEZB833 and IEEZB834 mapping macros in the program. IEEZB833 maps the IEEVARYD input including the IEEVARYD device array (VDEVARR). IEEZB834 maps the VARY device service results.
3. Obtain storage for the IEEVARYD input, IEEVARYD device array, and VARY device service results. An IEEVARYD device array entry is required for each device affected by a single invocation of IEEVARYD.
4. Initialize the IEEVARYD input for the operation by setting the appropriate keyword and option flags.
5. Initialize the IEEVARYD device array entries with the device number of each device upon which the operation is to be performed.
6. Issue the execute form of IEEVARYD, specifying the VARY device service input, IEEVARYD device array, VARY device service results, and the IEEVARYD parameter list which was defined when the list form of IEEVARYD was issued.
7. Examine the return code returned in register 15 to determine the overall result of the operation.
8. Use the IEEZB834 mapping macro to determine the results of the operation for each device in the IEEVARYD device array.
9. Free the storage for the IEEVARYD input, IEEVARYD device array, and VARY device service results.

```
*********************************************************************
*
*  Obtain storage for the IEEVARYD input, two IEEVARYD device array
*  Entries, and IEEVARYD Results
*  The address of the storage area is in R1
*
*  Initialize the IEEVARYD input
*
         LR    R2,R1              Address of storage
         LA    R4,0               Set to zero for MVCL
         LA    R5,0               Set to zero for MVCL
         MVCL  R2,R4              Clear storage
         USING VDEV,R2            Obtain addressability
         L     R0,CBID            Load identifier
         ST    R0,VDEV_ID         Initialize the identifier
         MVI   VDEV_VERSION,VDEV_VERN Initialize the version number
         OI    VDEV_KEYWORDS1,VDEV_ONLINE Initialize the operation to
*                                 ONLINE
         LA    R0,VDEV_LENGTH     Length of the IEEVARYD input
         LR    R4,R2              Address of the IEEVARYD input
         ALR   R4,R0              Address of storage immediately
*                                 following the IEEVARYD input
         LR    R5,R4              Save address of the
*                                 IEEVARYD device array
*
*  Initialize the first IEEVARYD device array entry
*
         USING VDEVARR,R4         Obtain addressability
         LA    R0,200
         STH   R0,VDEVARR_DEVN    Initialize the device number (200)
*
*  Initialize the second IEEVARYD device array entry
*
         LA    R0,VDEVARR_LENGTH  Length of IEEVARYD device array entry
         ALR   R4,R0              Obtain addressability to next entry
```

```
         LA    R0,300
         STH   R0,VDEVARR_DEVN    Initialize the device number (300)
*                                 following the IEEVARYD input
*********************************************************************
*
*  Get address of IEEVARYD Results
*
*********************************************************************
         ALR   R4,R0              Address of VDRSARR
*********************************************************************
*
*  Issue the execute form of IEEVARYD to vary the devices online
*
*********************************************************************
         IEEVARYD OPERATION=(R2),DEVICES=(R5),NUMDEVS=#DEVS,           X
               RESULTS=(R4),CALLERID=VDEVICES_ID,MF=(E,IEEVARYL)
*
*  Determine if the operation was not performed because of an error in
*  the parameters (Register 15 = 8)
*
         LA    R0,8
         CR    R15,R0
         BE    FREEVDEV           If the parameters are in error, free
*                                 the storage for the vary device array
*                                 header and vary device array entries
         USING VDRSARR,R4         Obtain addressability to RESULTS
         LA    R7,1               Initialize counter for loop
*
*  Perform the following loop for each IEEVARYD device array entry to
*  determine the results of the operation against each device
*
LOOP     DS    0H
         C     R7,#DEVS          All entries processed?
         BH    ENDLOOP
         TM    VDRSARR_OUTPUT_FLAGS1,VDRSARR_OUTPUT_VALID Determine if
*                                 output was returned for the device
         BZ    ITERATE            No output for the device, so iterate
         L     R9,VDRSARR_RETCODE Get return code for the device
         LA    R0,VDRSARR_ONLINE_WITH_REST Set register 0 to
*                                 VDRSARR_ONLINE_WITH_REST
         CR    R9,R0              Determine if the operation was
*                                 successful against the device
*                                 (VDRSARR_RETCODE <=
*                                 VDRSARR_ONLINE_WITH_REST)
         BH    BADRETC
GOODRETC DS    0H                 The operation was successful against
*                                 the device
         B     CHECKMSG
BADRETC  DS    0H                 The operation was not successful
*                                 against the device
CHECKMSG DS    0H                 Determine if a message was returned
*                                 for the device
         TM    VDRSARR_OUTPUT_FLAGS1,VDRSARR_MSG_RETURNED
         BZ    NOMSG
MSG      DS    0H                 A message was returned for the device
         B     ITERATE
NOMSG    DS    0H                 A message was not returned for the
*                                 device
ITERATE  DS    0H                 Prepare for the next iteration
         LA    R0,1
         ALR   R7,R0              Increment loop counter
         LA    R0,VDRSARR_LENGTH  Length of IEEVARYD device array entry
         ALR   R4,R0              Obtain addressability to next entry
         B     LOOP               Iterate
ENDLOOP  DS    0H                 End of loop
*
*  Release the storage for the IEEVARYD input, two
*  IEEVARYD device array entries, and IEEVARYD Results
*
FREEVDEV DS    0H
*********************************************************************
*                                                                   *
*  Declarations                                                     *
*                                                                   *
*********************************************************************
CBID     DC    C'VDEV'            Control block identifier for the
*                                 IEEVARYD input
VDEVICES_ID DC C'VDEVICES'        Caller identifier for the IEEVARYD
*                                 Input
@DATA    DS    0H
@DATD    DSECT
         DS    0F
```

```
#DEVS    DS    F
*************************************************************************
*
*  Issue the list form of IEEVARYD to define the parameter list
*
*************************************************************************
         IEEVARYD MF=(L,IEEVARYL)
@ENDDATD DS    0X
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
*************************************************************************
*
*  Include IEEZB833 to define the IEEVARYD input (VDEV) and
*  IEEVARYD device array entry (VDEVARR). Include IEEZB834
*  to define the IEEVARYD Results (VDRSARR).
*
*************************************************************************
         IEEZB833
         IEEZB834
         END   VDEVICES
```

IEEVARYD - List form

Use the list form of the IEEVARYD macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

The list form of the IEEVARYD macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEEVARYD.
IEEVARYD	
ƀ	One or more blanks must follow IEEVARYD.
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default:** 0D

The parameters are explained as follows:

MF=(L,*list addr*)
MF=(L,*list addr*,*attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IEEVARYD macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IEEVARYD - Execute form

Use the execute form of the IEEVARYD macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

The execute form of the IEEVARYD macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEEVARYD.
IEEVARYD	
ƀ	One or more blanks must follow IEEVARYD.
OPERATION=*operation parm*	*operation parm*: RS-type address or register (2) - (12).
,DEVICES=*devices parm*	*devices parm*: RS-type address or register (2) - (12).
,NUMDEVS=*num of devices*	*num of devices*: RS-type address or register (2) - (12).
,RESULTS=*vary results*	*vary results*: RS-type address or register (2) - (12).
,CALLERID=*caller id*	*caller id*: RS-type address or register (2) - (12).
,RETCODE=*return code*	*return code*: RS-type address or register (2) - (12).
,RSNCODE=*reason code*	*reason code*: RS-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).

Syntax	Description
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

The parameters are explained under the standard form of the IEEVARYD macro with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the IEEVARYD macro.

list addr specifies the area that the system uses to store the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that are not specified.

Chapter 87. IEFPPSCN — Scan the program properties table

Description

The IEFPPSCN macro provides a way to retrieve information, for report generation, about programs that are listed in the program properties table (PPT). IEFPPSCN allows the calling program to scan each entry in the PPT or to search the PPT for a specific program.

The installation controls what programs are listed in the PPT. An installation can specify a list of programs that require special attributes by using the SCHEDxx parmlib member with the PPT statement. The system then creates entries for these programs in the PPT. See *z/OS MVS Initialization and Tuning Reference* for information about using the SCHEDxx parmlib member.

The contents of the PPT can be dynamically changed through the SET command. Using the IEFPPSCN macro to retrieve information from the PPT prevents the system from dynamically updating the PPT while you are scanning it. If you scan the PPT without using IEFPPSCN, and the system updates the PPT while you are scanning it, your program will abnormally end.

Environment

Requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

The calling program must include the following mapping macros:

- CVT
- IEFJESCT
- IEFZB610

Restrictions

None.

Register information

After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the caller issued the macro. Therefore, if the caller

depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control is returned to the calling program the GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Return code

Performance implications

None.

Syntax

The standard form of the IEFPPSCN macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFPPSCN.
IEFPPSCN	
ƀ	One or more blanks must follow IEFPPSCN.
REQUEST=RETRIEVE	
REQUEST=NEXT	
REQUEST=END	
,PPTINFO=*ppt_info*	*ppt_info*: RX-type address or register (2) - (12).
	Required for REQUEST=RETRIEVE and REQUEST=NEXT. Not valid for REQUEST=END.
,PROGRAM=	*program_name*: RX-type address or register (2) - (12).
program_name	
	Required for REQUEST=RETRIEVE.
	Not valid for REQUEST=NEXT or REQUEST=END.

Syntax	Description
,TOKEN=*token*	*token*: RX-type address or register (2) - (12).
	Required for REQUEST=NEXT and REQUEST=END.
	Not valid for REQUEST=RETRIEVE.
,RETCODE=*rc*	*rc*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

REQUEST=RETRIEVE
REQUEST=NEXT
REQUEST=END
The required parameter that specifies what kind of request the caller is making.

Specify REQUEST=RETRIEVE to request information about a specific program. You must specify the name of the program on the PROGRAM parameter. You must also specify the PPTINFO parameter. Do not specify the TOKEN parameter.

If you want to scan all program entries sequentially, use the REQUEST=NEXT parameter together with the REQUEST=END parameter. Each time you specify REQUEST=NEXT, the system retrieves information about the next program entry. The first time you specify REQUEST=NEXT, you must put zero in the field you provide on the TOKEN parameter. On return to the caller, the system places a value in this field. After the first call, when you specify REQUEST=NEXT you must specify TOKEN and supply the value provided by the system on the previous call. With REQUEST=NEXT, you must also specify the PPTINFO parameter. Do not specify the PROGRAM parameter.

When control returns to the calling program with a return code of 4 in GPR 15, you have reached the end of the table and must then specify REQUEST=END. If you use REQUEST=NEXT and do not specify REQUEST=END, the system might not free common storage that could have been freed.

When you specify REQUEST=END, you must also specify the TOKEN parameter, supplying the value returned on the last REQUEST=NEXT. Do not specify the PROGRAM or PPTINFO parameters.

,PPTINFO=*ppt_info*
Specifies the area provided by the caller to contain the requested program information. The caller must provide this area as follows:

- The length of the area must be the length of the PPTENTRY field of the PPT plus two bytes. The PPT is mapped by the IEFZB610 mapping macro. See PPT in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the PPT mapping.
- Initialize the first two bytes of the area to the length of the PPTENTRY field.
- Define the length of the remainder of the area to be equal to the length of the PPTENTRY field.

Upon return, the system places the length of the requested PPTENTRY in the first two bytes of the area, and places the requested PPTENTRY itself in the remainder of the area.

This area must have a storage key that matches the PSW key of the issuer of IEFPPSCN. If the area provided is too small, the information is truncated. PPTINFO is required for REQUEST=RETRIEVE and REQUEST=NEXT. Do not specify PPTINFO with REQUEST=END.

,PROGRAM=*program_name*
Specifies the 8-character field containing the name of the program you want to retrieve when you specify REQUEST=RETRIEVE. If the program name is less than 8 characters, left justify the name and pad on the right with blanks.

Do not specify PROGRAM with REQUEST=NEXT or with REQUEST=END.

,TOKEN=*token*
Specifies the 4-byte field to contain the token that the system returns when you specify REQUEST=NEXT. Set the value of the token to zero before the first call. On subsequent calls made with REQUEST=NEXT or REQUEST=END, use the value of the token returned by the system on the previous REQUEST=NEXT.

Do not specify TOKEN with REQUEST=RETRIEVE.

,RETCODE=*rc*
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

Return codes

When control returns from IEFPPSCN, GPR 15 (and *rc*, if you coded RETCODE) contains one of the following return codes:

Table 81. Return Codes for the IEFPPSCN Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Program found. **Action**: No action required.
04	**Meaning**: End of table. **Action**: Issue IEFPPSCN with REQUEST=END.
08	**Meaning**: The program name you specified is not listed in the PPT, indicating that the installation did not specify any special attributes for that program. **Action**: No action required.
0C	**Meaning**: The token passed was not the token created by this macro. **Action**: Check that the application is coded to pass the correct token.
10	**Meaning**: Request not valid. **Action**: Check that you did not change the expanded assembler code.
14	**Meaning**: System error. The system was not able to obtain the required storage. Your program might issue a message indicating incomplete scanning of the PPT. **Action**: Reissue the request. If the error persists, contact your IBM support personnel.

Example

Sequentially scan the PPT and write the program name of each entry to the console. In your own code, you might wish to format the non-EBCDIC portions of the PPT entry for inclusion in the WTO.

The code in this example is nonreentrant. The caller is APF-authorized, and is initially in problem state with PSW key 8. The caller changes to supervisor state with PSW key 0 before issuing IEFPPSCN, and returns to problem state with PSW key 8 on completion of processing.

Note: This example is in SYS1.SAMPLIB in the member SHOWPPT.

```
         TITLE 'SHOWPPT - Show all entries in Current PPT'
SHOWPPT  CSECT                    Module entry point
SHOWPPT  AMODE 31
SHOWPPT  RMODE ANY
*
* Body of nonreentrant module which prints program names in the PPT
*
```

```
         STM   14,12,12(13)            Standard module linkage
         LR    12,15
         USING SHOWPPT,12
         ST    13,SAVEAREA+4
         LR    2,13
         LA    13,SAVEAREA
         ST    13,8(2)
*
         MODESET MODE=SUP,KEY=ZERO     Need supervisor state, key 0
*
* Set up for looking at PPT entries
*
         LA    2,PPTENT                Point to copy of PPT entry
         USING PPTENTRY,2              Set up addressability
         XC    SCNTOKEN,SCNTOKEN       Clear token
*
LOOP     DS    0H                      Loop getting PPT entries
         IEFPPSCN REQUEST=NEXT,TOKEN=SCNTOKEN,PPTINFO=PPTE
         LTR   15,15                   Check whether entry was returned
         BNZ   ENDLOOP
         MVC   TEXT2(8),PPTNAME        Copy program name to message
         WTO   TEXT=ENTRYTXT
         B     LOOP                    Get next entry, if any
*
ENDLOOP  DS    0H                      Finished with PPT
         IEFPPSCN REQUEST=END,TOKEN=SCNTOKEN
         MODESET MODE=PROB,KEY=NZERO   Return to problem state
*
*  Return to the calling program with the return code last passed by
*  IEFPPSCN.
*
         L     13,SAVEAREA+4           Return linkage
         L     14,12(13)
         LM    0,12,20(13)
         BR    14
*
****          Local storage definitions         ****
SCNTOKEN DC A(0)                       PPT scan token
PPTE     DC AL2(L'PPTENTRY)            Length of a PPT entry
PPTENT   DS CL(L'PPTENTRY)             PPT entry return area
SAVEAREA DC  18F'0'
*
*  The following areas are used to print the program name within the
*  PPT.  Additional formatting is required to make all the
*  information readable.
*
ENTRYTXT DS  0F                        Area for printing
TEXTL    DC  H'32'                     Message length
TEXT1    DC  C'SHOWPPT: Program Name = ' Constant portion of message
TEXT2    DS  CL8                       Variable portion (program name)
*
*  The following mapping macros are required for the IEFPPSCN macro.
*
         CVT DSECT=YES
         IEFJESCT
         IEFZB610
         END   SHOWPPT            End of SHOWPPT
```

IEFPPSCN - List form

Use the list form of the IEFPPSCN macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

The list form of the IEFPPSCN macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.

Syntax	Description
ƀ	One or more blanks must precede IEFPPSCN.
IEFPPSCN	
ƀ	One or more blanks must follow IEFPPSCN.
,MF=(L,*cntl)*	*cntl*: Symbol.
,MF=(L,*cntl,attr)*	*attr*: 1- to 60-character input string.
,MF=(L,*cntl*,0D)	**Default**: 0D.

Parameters

The parameters are explained under the standard form of the IEFPPSCN macro with the following exception:

,MF=(L,*cntl*)
,MF=(L,*cntl,attr*)
,MF=(L,*cntl*,0D)
Specifies the list form of the macro.

cntl is the name of a storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IEFPPSCN - Execute form

Use the execute form of the IEFPPSCN macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IEFPPSCN macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFPPSCN.
IEFPPSCN	
ƀ	One or more blanks must follow IEFPPSCN.

Syntax	Description
REQUEST=RETRIEVE	
REQUEST=NEXT	
REQUEST=END	
,PPTINFO=*ppt_info*	*ppt_info*: RX-type address or register (2) - (12).
	Required for REQUEST=RETRIEVE and REQUEST=NEXT.
	Not valid for REQUEST=END.
,PROGRAM=*program_name*	*program_name*: RX-type address or register (2) - (12).
	Required for REQUEST=RETRIEVE.
	Not valid for REQUEST=NEXT or REQUEST=END.
,TOKEN=*token*	*token*: RX-type address or register (2) - (12).
	Required for REQUEST=NEXT and REQUEST=END.
	Not valid for REQUEST=RETRIEVE.
,RETCODE=*rc*	*rc*: RX-type address or register (2) - (12).
,MF=(E,*cntl)*	*cntl*: RX-type address or register (2) - (12).
,MF=(E,*cntl*,COMPLETE)	**Default**: COMPLETE

Parameters

The parameters are explained under the standard form of the IEFPPSCN macro with the following exception:

,MF=(E,*cntl*)
,MF=(E,*cntl*,COMPLETE)
Specifies the execute form of the macro.

cntl is the name of a storage area for the parameter list.

COMPLETE specifies that the system is to check the macro parameter syntax and supply defaults on parameters that you do not use. COMPLETE is the default.

Chapter 88. IEFQMREQ — Invoke SWA manager in move mode

Description

Use this macro to read information from the SWA into a buffer that you provide, or to write information from a buffer into the SWA. *z/OS MVS Programming: Authorized Assembler Services Guide* describes how to use the IEFQMREQ macro.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the caller's primary address space

Programming requirements

The caller must include the following mapping macros:

- CVT
- IEFJESCT
- IEFQMIDS
- IEFQMNGR
- IEFZB506

Provide input to the IEFQMREQ macro through the external parameter area (EPAM), mapped by IEFZB506, and the queue manager parameter area (QMPA), mapped by IEFQMNGR.

Restrictions

None.

Input register information

On input to the macro, general purpose register (GPR) 1 must contain the address of the QMPA, and GPR 13 must contain the address of a standard 18-word save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
When control returns from IEFQMREQ, unchanged.

When control does not return from IEFQMREQ, the address of a 16-byte area containing:

Bytes 1-4
Address of the QMPA

Bytes 5-12
Not an intended programming interface; record this information and provide it to the appropriate IBM support personnel.

Bytes 13-16
Address of the failing EPA

1
When control returns from IEFQMREQ, used as a work register by the system.

When control does not return from IEFQMREQ, abend code 0B0.

2-14
Unchanged

15
Return code, when control returns from IEFQMREQ.

Reason code associated with the abend, when control does not return from IEFQMREQ.

Syntax

The IEFQMREQ macro, which has no parameters, is written as follows:

Syntax	Description
name	*name*:
ƀ	One or more blanks must precede IEFQMREQ.
IEFQMREQ	
ƀ	One or more blanks must follow IEFQMREQ.

Parameters

For information about initializing the parameter areas for IEFQMREQ, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

ABEND codes

The caller might encounter abend code X'0B0'

Return and reason codes

The hexadecimal return code is in GPR 15. When control returns from IEFQMREQ, the return codes have the following meanings.

Table 82. Return Codes for the IEFQMREQ Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: The IEFQMREQ service was successful.
38	**Meaning**: The system could not obtain the storage necessary to carry out the request.

When control does not return from IEFQMREQ, GPR 15 contains a hexadecimal reason code associated with system abend code 0B0. The reason codes have the following meanings.

Table 83. Reason Codes for the IEFQMREQ Macro

Hexadecimal Reason Code	Meaning
04	**Meaning**: The function you requested was not valid.
08	**Meaning**: The SVA in the SWA prefix was not valid.
0C	**Meaning**: You attempted to read a block that was not yet written.
10	**Meaning**: The length of an SWA block was not valid.
14	**Meaning**: The count field was not valid.
1C	**Meaning**: The block ID was not valid.
24	**Meaning**: The SVA does not correspond to any virtual address.

Chapter 89. IEFSJSYM — JCL symbol service

Description

The IEFSJSYM JCL symbol service provides JCL symbol information from the submitted JCL to the program running under the submitted JCL. The JCL symbol service performs the following functions:

REQUEST=GETALL
Returns all of the JCL symbols and values for the job step in the area provided by the caller, and is mapped by the IEFSJSYD macro.

REQUEST=GETBYNAME
Returns symbol values for the symbol names provided by the caller by the `SymListArray` parameter.

To be visible to the program, the symbols must have been either exported prior to the job step or provided by the submitter. The symbols are returned without a leading ampersand character (&).

The following information is described once at the beginning of the IEFSJSYM macro description:

- Environment
- Programming requirements
- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:

- Abend codes
- Return and reason codes
- Examples

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or Access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space

Programming requirements

REQUEST=GETBYNAME
The caller must provide a list of valid symbol names. Invalid symbol names or symbols that were not exported will have a null symbol value and a symbol value length of zero. A return code of 4 will be returned to indicate that not all symbols were processed successfully.

IEFSJSYD macro
To map data returned by an IEFSJSYM request.

Restrictions

This service cannot be used reliably until the job has begun execution. Invoking the service before the first job step has started executing is not supported (for example, in exits such as IEFUJI that are invoked before the first job step has started executing).

When using the returned symbol values, the value of the symbol returned will be the last value set prior to or within the current job step (EXEC PGM=statement).

Input register information

There are no input register requirements for issuing the IEFSJSYM macro.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

REQUEST= parameter of IEFSJSYM

The IEFSJSYM macro with the REQUEST parameter produces a DSECT that maps the format of the function routine input table.

Syntax

The syntax of the IEFSJSYM macro with REQUEST= is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSJSYM.
IEFSJSYM	
ƀ	One or more blanks must follow IEFSJSYM.
REQUEST=GETALL	Either REQUEST=GETALL or REQUEST=GETBYNAME is required.
REQUEST=GETBYNAME	
,SYMLISTARRAY=*symlistarray*	Required for REQUEST=GETBYNAME only.
,NUMENTRIES=*numentries*	Required for REQUEST=GETBYNAME only.
,SYMBAREA=*xsymbarea*	FSYM
,SYMBAREALEN=*symbarealen*	
,DIAGDATA=*diagdata*	
,RETCODE=*retcode*	
,RSNCODE=*rsncode*	
,PLISTVER=IMPLIED_VERSION	`IMPLIED_VERSION` is the default value.
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	S is the default value.
,MF=(L,*list addr*,0D)	`0D` is the default value.
,MF=(L,*list addr*,*attr*)	
,MF=(E,*list addr*,COMPLETE)	`COMPLETE` is the default value.
,MF=(E,*list addr*,NOCHECK)	

Syntax	Description
,MF=(M,*list addr*,COMPLETE)	COMPLETE is the default value.
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IEFSJSYM macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,DIAGDATA=*diagdata*
A required output parameter that specifies an area for service to return additional information. To code this parameter, specify an RS-type address, or address in register (2)-(12), of a 16-character field.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,list addr,COMPLETE)
,MF=(E,list addr,NOCHECK)
,MF=(M,list addr)
,MF=(M,list addr,COMPLETE)
,MF=(M,list addr,NOCHECK)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default value.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms of IEFSJSYM in the following order:

1. Use IEFSJSYM ...MF=(M,*list-addr*,COMPLETE) to specify appropriate parameters, including all required parameters.
2. Use IEFSJSYM ... MF=(M,*list-addr*,NOCHECK) to specify the parameters that you want to change.
3. Use IEFSJSYM ...MF=(E,*list-addr*,NOCHECK) to execute the macro.

,list addr
Specifies the name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1)-(12).

,attr
Specifies an optional 1-60 character input string which forces boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the

parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies for the system to check for required parameters, and to supply default values for omitted optional parameters.

,NOCHECK
Specifies for the system to not check for required parameters, and to not supply default values for omitted optional parameters.

,NUMENTRIES=*numentries*
A required input parameter for REQUEST=GETBYNAME that specifies the number of entries in the CHAR(16) array pointed to by SYMLISTARRAY. To code this parameter, specify an RS-type address or address in register (2)-(12) of a halfword field, or specify a literal decimal value.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
Specifies the version of the macro and determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using this parameter, specify it on all macro forms used for a request and with the same value on all of the macro forms. To code this parameter, specify IMPLIED_VERSION, MAX, or 0, as follows:

IMPLIED_VERSION
Specifies the lowest version that allows all of the specified parameters to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
Specifies to maximize the parameter list size. Because the supported maximum size can grow from release to release, the amount of storage that your program requires can also change. If your system can tolerate a size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all of the parameters that you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

0
Specifies to use the currently available parameters.

REQUEST=GETALL
REQUEST=GETBYNAME
A required parameter that specifies the JCL symbols to get. Use REQUEST=GETALL to get all symbol values that were exported. Use REQUEST=GETBYNAME to get specific named symbol values, given an array of symbol names.

,RETCODE=*retcode*
An optional output parameter into which the return code is copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15. To code this parameter, specify an RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0. To code this parameter, specify an RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).

,SYMBAREA=*xsymbarea*
The name (RS-type), or address in register (2)-(12), of a required character output storage area in the user's key which will contain all of the requested JCL-defined symbols and their values. This area is mapped by IEFSJSYD.

,SYMBAREALEN=*symbarealen*
A required input parameter that specifies the length of the SYMBAREA that is provided by the caller. To code this parameter, specify an RS-type address, or address in register (2)-(12) of a fullword field, or specify a literal decimal value.

,SYMLISTARRAY=*symlistarray*
A required input parameter for REQUEST=GETBYNAME that contains an array of up to 16 character entries, each of which contains a symbol name for which the symbol value is to be returned. Symbol names must be left-justified in the array entry, and if shorter than 16 characters, padded on the right with blank spaces.

A wildcard character (asterisk (*) to match 0 or more characters in the symbol name, or question mark (?) to match exactly one character) can used to specify a generic symbol name. Symbol names should not contain a leading ampersand character (&) or any special character other than a wildcard character.

IEFSJSYM returns a null value (SYDESYMVALUELEN=0) for entries that do not contain valid JCL symbol name; in addition, a return code of IEFSJSYMRC_Warn and a reason code of IEFSJSYMRsn_SymbolNameNotProcessed are set. To code this parameter, specify an RS-type address, or address in register (2)-(12), of a character field.

ABEND codes

None.

Return and reason codes

The following table, Table 84 on page 874, contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 84. Return and reason codes for the IEFSJSYM macro

Return Code	Reason Code	Equate Symbol for Reason Code Meaning and Action
00000004	xxxx0404	**Equate Symbol**: IEFSJSYMRsn_SymbolNameNotProcessed **Meaning**: For a GETBYNAME request, the values of some of the requested symbols could not be returned. This can occur if the symbol was not exported by the calling JCL, if the symbol was not SET after being exported, or if the symbol name input to IEFSJSYM did not follow JCL symbol name conventions. This only occurs for specific symbol names in the array, and not to symbol names that include wildcard characters. Symbols without values are returned with null values and symbol value lengths of 0. **Action**: Check the submitted JCL to ensure that an EXPORT was done for the requested symbol and that the symbol was SET after the EXPORT statement.
00000004	xxxx0408	**Equate Symbol**: IEFSJSYMRsn_InsSuffSymSpace **Meaning**: Insufficient space to return all of the symbols and values requested. **Action**: Use the value returned in SYDALEN to obtain the storage required to fit all of the returned symbols, values and control information.
00000008	xxxx0804	**Equate Symbol**: IEFSJSYMRsn_ParmlistAddrInvalid **Meaning**: IEFSJSYM could not use the parameter list provided. **Action**: Verify that the address of the parameter list is valid and resides in virtual storage of the primary address space.
00000008	xxxx0808	**Equate Symbol**: IEFSJSYMRsn_SymbareaAddrInvalid **Meaning**: IEFSJSYM could not use the output symbol area provided **Action**: Verify that the address of the symbol area is valid and resides in virtual storage of the primary address space.

Table 84. Return and reason codes for the IEFSJSYM macro (continued)

Return Code	Reason Code	Equate Symbol for Reason Code Meaning and Action
00000008	xxxx080C	**Equate Symbol**: IEFSJSYMRsn_SymbListAddrInvalid **Meaning**: IEFSJSYM could not use the input symbol list provided. **Action**: Verify that the address of the data area is valid and resides in virtual storage of the primary address space.
00000008	xxxx0810	**Equate Symbol**: IEFSJSYMRsn_Mismatched_VersLen **Meaning**: The length of the IEFSJSYM parameter list does not match the version number supplied. **Action**: Ensure that the parameter list that was built matches the specified or default parameter list version.
00000008	xxxx0814	**Equate Symbol**: IEFSJSYMRsn_Unsupported_version **Meaning**: The version of the parameter list is not supported with this level of the IEFSJSYM service. **Action**: Correct the version and other parameters to match the system where the job was run, or run the job on a system that supports this version of IEFSJSYM.
00000008	xxxx0818	**Equate Symbol**: IEFSJSYMRsn_Unsupported_Function **Meaning**: The request was for a function that is not supported with this level of the IEFSJSYM service. **Action**: Choose a function that is supported on this level of IEFSJSYM service, or request this function on a system that supports it.
0000000C	xxxx0C04	**Equate Symbol**: IEFSJSYMRsn_InsSuffHdSpace **Meaning**: Insufficient space to return the header (SYDHDR) portion of the data area. **Action**: Refer to the mapping macro IEFSJSYD and pass an area at least as large as the DSECT SYDHDR.
0000000C	xxxx0C08	**Equate Symbol**: IEFSJSYMRsn_StorageNotObtained **Meaning**: Failed to obtain above the bar storage via IARV64. The length of the storage requested is based on the size of the caller's SYMBAREA and SYMLISTARRAY size. **Action**: Check the SYMBAREALEN specification. If SYMBAREALEN is coded as an extremely large number, try reducing the SYMBAREALEN to a size that is comparable to the number of symbols requested.
0000000C	xxxx0C0C	**Equate Symbol**: IEFSJSYMRsn_IncorrectExecEnv **Meaning**: A proper execution environment does not exist for the service. **Action**: Verify that the program meets the requirements described previously. The returned DIAGDATA value will contain information that identifies the problem.
0000000C	xxxx0C10	**Equate Symbol**: IEFSJSYMRsn_UnexpectedSjfResponse **Meaning**: Underlying JCL services invoked by the service returned with a unexpected return and reason codes. IEFSJSYM might have been invoked before the job execution environment was established, or after the job execution environment had ended. **Action**: Verify that the program is running under a batch program environment. The returned DIAGDATA value contains information, such as the JCL service and return and reason codes, to diagnose the error.

Example

```
SJSYM_RC     DS   F
SJSYM_RSN    DS   F
SJDIAG       DS   4F
SYMBOLS      DS   0D
S1           DC   C'DSN     '
```

```
S2          DC   C'VOL     '
S3          DC   C'UNIT    '
SYMBOLAREA  DS   64F
  IEFSJSYM REQUEST=GETBYNAME,
     SYMLISTARRAY=SYMBOLS,NUMENTRIES=3,
     SYMBAREA=SYMBOLAREA,SYMBAREALEN=512,
     DIAGDATA=SJDIAG
```

Chapter 90. IEFSSI — Dynamically control a subsystem

Description

Use the IEFSSI macro to dynamically control a subsystem in any of the following ways:

- Adding and defining a subsystem to the system
- Activating a subsystem so that its function routines can process function requests
- Defining a set of optional subsystem characteristics
- Deactivating a subsystem
- Swapping the current SSVT with a new SSVT
- Storing subsystem-defined data for a subsystem
- Retrieving subsystem-defined data for a subsystem that was previously stored with the put request
- Query information for all subsystems defined to the SSI

The requests for the macro are:

- IEFSSI REQUEST=ADD, dynamically adds and defines a subsystem to the system. See "REQUEST=ADD parameter of IEFSSI" on page 879 for the syntax of this request.
- IEFSSI REQUEST=ACTIVATE, dynamically activates a subsystem so that its function routines are available to process function requests. See "REQUEST=ACTIVATE parameter of IEFSSI" on page 883 for the syntax of this request.
- IEFSSI REQUEST=OPTIONS, which defines a set of optional subsystem characteristics. See "REQUEST=OPTIONS parameter of IEFSSI" on page 886 for the syntax of this request.
- IEFSSI REQUEST=DEACTIVATE, which deactivates a subsystem. See "REQUEST=DEACTIVATE parameter of IEFSSI" on page 889 for the syntax of this request.
- IEFSSI REQUEST=SWAP, which replaces the SSVT that is currently being used to route function requests with a new one. See "REQUEST=SWAP parameter of IEFSSI" on page 892 for the syntax of this request.
- IEFSSI REQUEST=PUT, which stores subsystem-defined data for the subsystem. See "REQUEST=PUT parameter of IEFSSI" on page 895 for the syntax of this request.
- IEFSSI REQUEST=GET, which retrieves subsystem-defined data previously stored using the IEFSSI REQUEST=PUT service. See "REQUEST=GET parameter of IEFSSI" on page 898 for the syntax of this request.
- IEFSSI REQUEST=QUERY, which obtains information about a currently defined subsystem. See "REQUEST=QUERY parameter of IEFSSI" on page 901 for the syntax of this request.

The IEFSSI macro (REQUEST=QUERY only) is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*.

For ease of use, the standard form of the macro is shown for each IEFSSI request. The requests are described on the following pages along with the:

- Standard form syntax diagram
- Description of the parameters

The following information is described once at the beginning of the IEFSSI macro description:

- Environment
- Programming requirements

- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:

- Abend codes
- Return and reason codes
- Examples

The REQUEST=ADD, REQUEST=ACTIVATE, REQUEST=OPTIONS, REQUEST=DEACTIVATE, REQUEST=SWAP, REQUEST=PUT, REQUEST=GET and REQUEST=QUERY parameters, which designate the services of the IEFSSI macro, are mutually exclusive. You can select only one.

For information about using dynamic subsystem services, see *z/OS MVS Using the Subsystem Interface*. This topic also includes information about related macros IEFSSVT and IEFSSVTI.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For the QUERY request, problem state with any PSW key. The REQUEST=ADD, REQUEST=ACTIVATE, REQUEST=OPTIONS, REQUEST=DEACTIVATE, REQUEST=SWAP, REQUEST=PUT, and REQUEST=GET parameters require one of the following: • Supervisor state • Any system PSW key • PSW key mask (PKM) allowing key 0-7 • APF authorization
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24-bit or 31-bit
ASC mode:	Primary or Access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space

Programming requirements

- Include the CVT and IEFJESCT mapping macros in your program.
- Include the IEFJSRC mapping macro in your program. This macro defines the dynamic SSI return and reason codes.
- For the REQUEST=QUERY parameter, the caller must include the IEFJSQRY macro to map the REQUEST=QUERY output.
- For the REQUEST=ACTIVATE and REQUEST=SWAP parameters, the subsystem must have created at least one SSI-managed vector table. An SSI-managed vector table is a vector table created with the IEFSSVT REQUEST=CREATE macro.

Restrictions

The caller must not have established an EUT FRR.

Input register information

Before issuing the IEFSSI macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2 - 13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0 - 1
Used as a work register by the system.

2 - 13
Unchanged

14 - 15
Used as a work register by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

REQUEST=ADD parameter of IEFSSI

The IEFSSI macro with the ADD parameter dynamically adds and defines a subsystem to the system.

Syntax for REQUEST=ADD

The syntax of the IEFSSI REQUEST=ADD macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.

Syntax	Description
␣	One or more blanks must precede IEFSSI.
IEFSSI	
␣	One or more blanks must follow IEFSSI.
SUBNAME=*subname*	*subname:* RS-type address or address in register (2) - (12).
,REQUEST=ADD	
,CONSNAME=*consname*	*consname:* RS-type address or address in register (2) - (12).
,CONSNAME=0	**Default**: CONSNAME=0
,INITRTN=*initrtn*	*initrtn:* RS-type address or address in register (2) - (12).
,INITRTN=NO_INITRTN	**Default**: INITRTN=NO_INITRTN
,INITPARM=*initparm*	*initparm:* RS-type address or address in register (2) - (12).
,INITPARM=NO_INITPARM	**Default**: INITPARM=NO_INITPARM
,INITPLEN=*initplen*	*initplen*: RS-type address or address in register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	

Syntax	Description
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=ADD

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name.

This fullword field must be padded to the right with blanks or nulls if it is less than 4 characters long.

When selecting subsystem names, note the following:

- If you specify a subsystem name with the characters '*' and '?', the DISPLAY SSI command or the IEFSSI REQUEST=QUERY service specifying that subsystem name may return information about subsystems other than this one. The '*' and '?' are treated as wildcard characters for these services.
- If you specify a subsystem name of '!PRI', the DISPLAY SSI command or the IEFSSI REQUEST=QUERY service specifying that subsystem name returns information about the primary subsystem, even though there is a subsystem named '!PRI'.

Note: If you need to start the subsystem, its name must meet the requirements for the name of a started task. See *z/OS MVS JCL Reference* for more information.

,REQUEST=ADD
A parameter that specifies that a subsystem is to be dynamically defined.

,CONSNAME=*consname*
,CONSNAME=0
An optional 8-character parameter that specifies the name (or an address in a register) of a console to which any messages the SSI issues as part of the initialization process are written. If an INITRTN parameter is specified, the console name is passed to the routine named on INITRTN.

The default is 0. If the default parameter is used, the SSI issues messages to the consoles that are receiving the INTIDS attribute.

,INITRTN=*initrtn*
,INITRTN=NO_INITRTN
An optional 8-character parameter that specifies the name (or an address in a register) of a subsystem initialization routine.

A subsystem initialization routine name that is less than 8 characters long must be padded to the right with blanks. The default is NO_INITRTN.

,INITPARM=*initparm*
,INITPARM=NO_INITPARM
An optional parameter that specifies the name (or address in a register) of a parameter string that is passed to the subsystem initialization routine. This parameter string can be up to 60 characters long. The INITPLEN parameter specifies the actual length of the passed parameter.

The INITPARM parameter is applicable only if you specify the INITRTN parameter.

,INITPLEN=*initplen*
A required parameter that contains the length of the parameter string to be passed to the subsystem initialization routine. You must specify this 4-byte parameter if you specify the INITPARM parameter.

INITPLEN can be from 1 to 60 characters long inclusive. If the length is greater than 60, the subsystem is defined but the subsystem initialization routine is not invoked.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=ACTIVATE parameter of IEFSSI

The IEFSSI macro with the ACTIVATE parameter dynamically activates a subsystem so that its function routines are available to process function requests.

Syntax for REQUEST=ACTIVATE

The syntax of the IEFSSI REQUEST=ACTIVATE macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEFSSI.
IEFSSI	
␣	One or more blanks must follow IEFSSI.
SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12).
,REQUEST=ACTIVATE	
,INTOKEN=*intoken*	*intoken*: RS-type address or address in register (2) - (12).
,INTOKEN=NO_INPUT_TOKEN	**Default**: INTOKEN=NO_INPUT_TOKEN
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable

Syntax	Description
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=ACTIVATE

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

,REQUEST=ACTIVATE
A parameter that specifies a subsystem is to be dynamically activated so that its function routines are available to process function requests. Before invoking the IEFSSI macro and issuing the REQUEST=ACTIVATE parameter, the subsystem must be defined to the system, and you must ensure that an SSVT has been built using the IEFSSVT macro with the REQUEST=CREATE parameter.

The ACTIVATE request may also be used to reactivate a subsystem that has been deactivated. To reactivate a subsystem, you can either use the same SSVT as you used to deactivate the subsystem or you can use a new SSVT.

,INTOKEN=*intoken*
,INTOKEN=NO_INPUT_TOKEN
An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the SSVT that is used when activating the subsystem. The function routines associated with the SSVT are made available for processing requests.

The token must be one that was returned by one of the following:

- IEFSSVT REQUEST=CREATE
- IEFSSI REQUEST=DEACTIVATE
- IEFSSI REQUEST=SWAP

If the INTOKEN parameter is omitted, an SSVT is chosen as follows:

- The most recently active SSI-managed vector table
- The last SSI-managed vector table created, if no SSI-managed vector table has been activated

The default is NO_INPUT_TOKEN.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=OPTIONS parameter of IEFSSI

The IEFSSI macro with the OPTIONS parameter defines a set of optional subsystem characteristics.

Syntax for REQUEST=OPTIONS

The syntax of the IEFSSI REQUEST=OPTIONS macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEFSSI.
IEFSSI	
␣	One or more blanks must follow IEFSSI.
SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12) of fullword output variable
,REQUEST=OPTIONS	
,COMMAND=NO	**Default**: COMMAND=NO
,COMMAND=YES	
,REQDSUB=MSTR	**Default**: REQDSUB=MSTR
,REQDSUB=PRI	
,EVENTRTN=*exitname*	*exitname*: 8-character module name
,EVENTRTN=NO_EVENTRTN	**Default**: EVENTRTN=NO_EVENTRTN
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1

Syntax	Description
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=OPTIONS

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

,REQUEST=OPTIONS
A parameter that specifies the definition of a set of optional subsystem characteristics. You can set the following subsystem options using this macro:

- Whether the subsystem responds to SETSSI commands
- Whether you want the invoking subsystem to start under the MSTR or primary subsystem

IEFSSI REQUEST=OPTIONS is the only way you can specify these optional characteristics.

If you invoke the OPTIONS parameter multiple times for a single subsystem, the most recent invocation determines the resulting characteristics. The defaults listed for the COMMAND and REQDSUB parameters below apply to the first invocation.

If a parameter is not specified on a subsequent invocation, the corresponding subsystem characteristic retains the value that was assigned by the last invocation that specified the parameter.

,COMMAND=NO
,COMMAND=YES
An optional parameter that specifies the whether the subsystem responds to the following commands:

- SETSSI ACTIVATE
- SETSSI DEACTIVATE

The meanings are:

NO
The subsystem does not allow SETSSI commands. NO is the default.

YES
The subsystem allows SETSSI commands.

You need to specify COMMAND=YES only if the subsystem can tolerate the processing associated with each of the SETSSI commands listed above.

,REQDSUB=MSTR
,REQDSUB=PRI
An optional parameter that specifies whether a START subsystem command causes the subsystem to start under either the MSTR subsystem or the primary subsystem (JES).

When the procedure name on the START command matches a defined subsystem name, the procedure being started is recognized as a subsystem. If the START command does not specify the SUB parameter, the subsystem is started under the control of the subsystem identified by the REQDSUB parameter.

The meanings are:

REQDSUB=MSTR
The subsystem specified does not require the services of the primary subsystem and starts under the MSTR subsystem. MSTR is the default.

REQDSUB=PRI
The subsystem specified requires the services of the primary subsystem and must start under its control. If a START subsystem command is issued before the primary subsystem is available, the processing that the subsystem was doing in response to the START command fails.

,EVENTRTN=*exitname*
,EVENTRTN=NO_EVENTRTN
An optional input parameter that specifies the name of the subsystem event notification routine to get control whenever a subsystem event occurs.

EVENTRTN=*exitname*
Specifies the 8-character module name (*exitname*), left-justified and padded on the right with blanks, if necessary.

EVENTRTN=NO_EVENTRTN
Indicates that there is no subsystem event notification routine. This will disable the current subsystem event notification routine, if there is one.

Default: EVENTRTN=NO_EVENTRTN

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=DEACTIVATE parameter of IEFSSI

The IEFSSI macro with the DEACTIVATE parameter deactivates a subsystem.

Syntax for REQUEST=DEACTIVATE

The syntax of the IEFSSI REQUEST=DEACTIVATE macro is written as follows:

Syntax	Description

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEFSSI.
IEFSSI	
␣	One or more blanks must follow IEFSSI.
SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12).
,REQUEST=DEACTIVATE	
,OUTTOKEN=*outtoken*	*outtoken*: RS-type address or address in register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=DEACTIVATE

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

,REQUEST=DEACTIVATE
A parameter that specifies that a subsystem is to be deactivated. This stops any new function requests from being passed to the subsystem function routines.

A subsystem can be reactivated after being deactivated by using the same or a different SSVT.

,OUTTOKEN=*outtoken*
An optional 32-bit parameter that specifies the name (or an address in a register) of an output token. This is where the token that represents the deactivated SSVT is returned.

This token may be used in a subsequent ACTIVATE request to reactivate the subsystem using the same SSVT.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=SWAP parameter of IEFSSI

The IEFSSI macro with the SWAP parameter replaces the SSVT that is currently being used to route function requests with a new one.

Syntax for REQUEST=SWAP

The syntax of the IEFSSI REQUEST=SWAP macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEFSSI.
IEFSSI	
␣	One or more blanks must follow IEFSSI.

Syntax	Description
SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12).
,REQUEST=SWAP	
,INTOKEN=*intoken*	*intoken*: RS-type address or address in register (2) - (12).
,INTOKEN=NO_INPUT_TOKEN	**Default**: INTOKEN=NO_INPUT_TOKEN
,OUTTOKEN=*outtoken*	*outtoken*: RS-type address or address in register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=SWAP

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

,REQUEST=SWAP
A parameter that specifies the replacement of the SSVT currently being used to route function requests with a new SSVT. The current SSVT is deactivated and the new SSVT is (re)activated. The subsystem remains continuously active during this process.

When you use a SWAP request to switch SSVTs, it is possible for you to use a subsequent SWAP request to switch the SSVTs again, which restores the old function routines.

A SWAP request that targets an inactive subsystem is treated as an ACTIVATE request, but receives the IEFSSI_WARNING (4) return code.

,INTOKEN=*intoken*
,INTOKEN=*NO_INPUT_TOKEN*
An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the SSVT that is used when activating the subsystem. The function routines associated with the SSVT are made available for processing requests.

The token must be one that was returned by either the:

- IEFSSVT REQUEST=CREATE
- IEFSSI REQUEST=DEACTIVATE
- IEFSSI REQUEST=SWAP

If the INTOKEN parameter is omitted, an SSVT is chosen as follows:

- The most recently active SSI-managed vector table
- The last SSI-managed vector table created, if no SSI-managed vector table has been activated

The default is NO_INPUT_TOKEN.

,OUTTOKEN=*outtoken*
An optional 32-bit parameter that specifies the name (or an address in a register) of an output token. This is where the token that represents the deactivated SSVT is returned.

This token may be used in a subsequent SWAP request to reactivate the subsystem using the same SSVT.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX

- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=PUT parameter of IEFSSI

The IEFSSI macro with the PUT parameter stores subsystem-defined data for the subsystem.

Syntax for REQUEST=PUT

The syntax of the IEFSSI REQUEST=PUT macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEFSSI.

Syntax	Description
IEFSSI	
␣	One or more blanks must follow IEFSSI.
SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12).
,REQUEST=PUT	
,SUBDATA1=*subdata1*	*subdata1*: RS-type address or address in register (2) - (12). of a 4-character input area
,SUBDATA2=*subdata2*	*subdata2*: RS-type address or address in register (2) - (12). of a 4-character input area
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable.
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=PUT

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

,REQUEST=PUT
A parameter that specifies the storage of subsystem-defined data for the subsystem. Two non-contiguous 4-byte fields are available for the subsystem data. One of these fields is typically used to anchor subsystem specific control blocks.

You must specify at least one of the following parameters:

SUBDATA1=*subdata1*
The name (or address in a register) of a 4-character input area that holds the first 4-bytes of subsystem specific information.

SUBDATA2=*subdata2*
The name (or address in a register) of a 4-character input area that holds the second 4-bytes of subsystem specific information.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=GET parameter of IEFSSI

The IEFSSI macro with the GET parameter retrieves subsystem-defined data previously stored using the IEFSSI REQUEST=PUT request.

Syntax for REQUEST=GET

The syntax of the IEFSSI REQUEST=GET macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEFSSI.
IEFSSI	
␣	One or more blanks must follow IEFSSI.
SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12).
,REQUEST=GET	

Syntax	Description
,SUBDATA1=*subdata1*	*subdata1*: RS-type address or address in register (2) - (12) of a 4-character output area.
,SUBDATA2=*subdata2*	*subdata2*: RS-type address or address in register (2) - (12) of a 4-character output area.
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable.
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=GET

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

,REQUEST=GET
A parameter that specifies the retrieval of subsystem-defined data previously stored using the IEFSSI REQUEST=PUT request. Two non-contiguous 4-byte fields are available for the subsystem data.

You must specify at least one of the following parameters:

SUBDATA1=*subdata1*
The name (or address in a register) of a 4-character output area that holds the first 4-bytes of subsystem specific information identified by the SUBDATA1 parameter on a previous IEFSSI REQUEST=PUT request.

SUBDATA2=*subdata2*
The name (or address in a register) of a 4-character output area that holds the second 4-bytes of subsystem specific information identified by the SUBDATA2 parameter on a previous IEFSSI REQUEST=PUT request.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=QUERY parameter of IEFSSI

The IEFSSI macro with the QUERY parameter requests information about subsystems defined to the system.

Syntax for REQUEST=QUERY

The syntax of the IEFSSI REQUEST=QUERY macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IEFSSI.
IEFSSI	
␣	One or more blanks must follow IEFSSI.
SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12).
,REQUEST=QUERY	
,WORKAREA=*workarea*	*workarea*: RS-type address or address in register (2) - (12) of an output area.
,WORKASP=*workasp*	*workasp*: RS-type address or address in register (2) - (12) of an input area.

Syntax	Description
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12). of fullword output variable
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12). of fullword output variable
,COM=*com*	*com*: comment string
,COM=NULL	**Default**: COM=NULL
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters for REQUEST=QUERY

The parameters are explained as follows:

SUBNAME=*subname*
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

For the REQUEST=QUERY parameter, the subsystem name may contain the wildcard characters '*' and '?' to request information about multiple subsystems. The meanings for the wildcard characters are:

*****
Matches 0 or more characters.

Use a SUBNAME parameter value of '*' to indicate that information is to be returned for all subsystems.

?
Matches exactly 1 character

Use a SUBNAME parameter value of '!PRI' to indicate that information is to be returned for the primary subsystem.

,REQUEST=QUERY
A parameter that specifies the request to obtain information about a currently defined subsystem named in the SUBNAME parameter.

The output from IEFSSI REQUEST=QUERY is mapped by the IEFJSQRY macro. Subsystems are listed in broadcast order, that is, the order in which they receive broadcast SSI requests.

,WORKAREA=*workarea*
A required parameter that specifies a name (or register containing the address) of a pointer output field that contains the address of the subsystem information returned by the QUERY request.

The output area is mapped by the IEFJSQRY macro. The JQRYLEN field contains the length of the output area.

,WORKASP=*workasp*
An optional parameter that specifies a name (or register containing the address) of a one-byte input field that specifies the subpool that the SSI uses to obtain a work area for the returned subsystem information. The caller is responsible for freeing this work area.

IBM recommends that you use a job-related or task-related subpool. This allows the system to free the associated storage when the job or task ends, if the caller does not free the returned area.

If WORKASP is not specified, the caller's subpool zero is used. Storage for the query information is obtained above 16 megabytes. AMODE 24 callers must switch into AMODE 31 to address this storage. Unauthorized callers may request storage only in the following unauthorized subpools:

- 0-127
- 131
- 132

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies the name of an output field (or a register) where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=NULL
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSI macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSI macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSI macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1- to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

ABEND codes

An invocation of the IEFSSI macro may result in an abend code X'8C5'. See *z/OS MVS System Codes* for an explanation of this abend code.

Return and reason codes

When the IEFSSI macro returns control to your program, GPR 15 (and *retcode*, if you coded RETCODE) contains a return code. When the value in GPR 15 is not 0, GPR 0 (and *rsncode* if you coded RSNCODE) contains the reason code.

The IEFJSRC mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each return code are:

Decimal (Hex)
Equate Symbols

00 (00)
IEFSSI_SUCCESS

04 (04)
IEFSSI_WARNING

08 (08)
IEFSSI_INVALID_PARAMETERS

12 (0C)
IEFSSI_REQUEST_FAIL

20 (14)
IEFSSI_SYSTEM_ERROR

24 (18)
IEFSSI_UNAVAILABLE

The following table contains return and reason codes, the equate symbols associated with each reason code and the meaning and suggested action for each return and reason code.

Table 85. Return and reason codes for the IEFSSI macro

Return code decimal (hex)	Reason code decimal (hex)	Meaning and action
00 (00)	00 (00)	**Equate symbol**: IEFSSI_FUNCTIONS_COMPLETE **Meaning**: The request completed successfully. The result depends on the request: • ADD — A subsystem has been added to the system • ACTIVATE — A subsystem has been activated • OPTIONS — A set of optional subsystem characteristics has been defined • DEACTIVATE — A subsystem has been deactivated • SWAP — The current SSVT has been swapped with a new SSVT • PUT — Subsystem-defined data has been stored • GET — Subsystem-defined data has been retrieved • QUERY — Information for all subsystems defined to the SSI has been queried **Action**: None.
04 (04)	300 (12C)	**Equate symbol**: IEFSSI_DEACT_INACTIVE **Meaning**: The subsystem was already inactive. This is a DEACTIVATE request error. **Action**: None.
04 (04)	301 (12D)	**Equate symbol**: IEFSSI_DEACT_OUT_VT_NOT_SSI **Meaning**: The subsystem is deactivated, however a previously active vector table was not SSI-managed. OUTTOKEN value is 0. This is a DEACTIVATE request error. **Action**: None.
04 (04)	500 (1F4)	**Equate symbol**: IEFSSI_SWAP_INACTIVE **Meaning**: The subsystem was initially active. OUTTOKEN value is 0. This is a SWAP request error. **Action**: None.
04 (04)	501 (1F5)	**Equate symbol**: IEFSSI_SWAP_OUT_VT_NOT_SSI **Meaning**: The swap is complete, however the previously active vector table was not SSI-managed. OUTTOKEN value is 0. This is a SWAP request error. **Action**: None.
04 (04)	900 (384)	**Equate symbol**: IEFSSI_QUERY_INCOMPLETE **Meaning**: The data returned by the QUERY request may be incomplete. This is a QUERY request error. **Action**: Check the JQRY_INCOMPLETE flag for each subsystem that was queried.

Table 85. Return and reason codes for the IEFSSI macro (continued)

Return code decimal (hex)	Reason code decimal (hex)	Meaning and action
08 (08)	00 (000)	**Equate symbol**: IEFSSI_SUBSYSTEM_UNKNOWN **Meaning**: The subsystem is not defined to the SSI. **Action**: Correct the subsystem name or define a subsystem with either the IEFSSI macro or the SETSSI command.
08 (08)	04 (004)	**Equate symbol**: IEFSSI_NON_DYNAMIC **Meaning**: The subsystem is not dynamic. **Action**: ReIPL the system and define the target subsystem with either the IEFSSI macro, the SETSSI command, or the keyword format IEFSSNxx parmlib member entry. Note that once a subsystem has been defined, it cannot be deleted or defined again as dynamic.
08 (08)	08 (008)	**Equate symbol**: IEFSSI_BAD_VT_TOKEN **Meaning**: The SSVT token does not correspond to a valid SSVT table. **Action**: Correct the token. The token must be one that was returned by either the IEFSSVT REQUEST=CREATE macro, the IEFSSI REQUEST=DEACTIVATE macro, or the IEFSSI REQUEST=SWAP macro.
08 (08)	12 (00C)	**Equate symbol**: IEFSSI_INVALID_NAME **Meaning**: The subsystem name or the routine name contains characters that are not valid. **Action**: Correct the subsystem name by removing the characters that are not valid.
08 (08)	16 (010)	**Equate symbol**: IEFSSVT_INIT_PARMS **Meaning**: The initialization routine parameter string is too long. **Action**: Correct the parameter string so that it is no longer than 60 characters.
12 (0C)	100 (064)	**Equate symbol**: IEFSSI_DUPLICATE_SUBSYSTEM **Meaning**: The subsystem already exists. This is an ADD request error. **Action**: Do not perform the ADD request if the existing subsystem is one you want. If the existing subsystem is not the one you want, select another name for the new subsystem, which does not conflict with the name of any existing subsystem name. You can use the IEFSSI REQUEST=QUERY macro to find all existing names.
12 (0C)	101 (065)	**Equate symbol**: IEFSSI_INITRTN_NOT_FOUND **Meaning**: A usable copy of this initialization routine could not be found. This is an ADD request error. For example: • The module was not found. • The module was found, but was not APF-authorized. **Action**: Correct the initialization routine name or make sure it is present in either LINKLIB or LPALIB, and is APF authorized.
12 (0C)	102 (066)	**Equate symbol**: IEFSSI_INITRTN_ABEND **Meaning**: The initialization routine ended abnormally. This is an ADD request error. **Action**: Check the dump produced by the abend and correct the problem with the initialization routine.

Table 85. Return and reason codes for the IEFSSI macro (continued)

Return code decimal (hex)	Reason code decimal (hex)	Meaning and action
12 (0C)	103 (067)	**Equate symbol**: IEFSSI_ADD_STORAGE **Meaning**: Unable to obtain storage for the subsystem definition. This is an ADD request error. **Action**: Check the current use of the system storage to determine why storage was not available. Retry the request later in case storage has become available.
12 (0C)	200 (0C8)	**Equate symbol**: IEFSSI_SUBSYSTEM_ACTIVE **Meaning**: The subsystem is already active. This is an ACTIVATE request error. **Action**: None.
12 (0C)	201 (0C9)	**Equate symbol**: IEFSSI_ACT_NO_ELIGIBLE_VT **Meaning**: The SSVT is not specified and a valid default is not available. This is an ACTIVATE request error. **Action**: Provide an SSI-managed SSVT using the IEFSSVT REQUEST=CREATE macro.
12 (0C)	500 (1F4)	**Equate symbol**: IEFSSI_SWAP_NO_ELIGIBLE_VT **Meaning**: The SSVT is not specified and a valid default is not available. This is a SWAP request error. **Action**: Provide an SSI-managed SSVT using the IEFSSVT REQUEST=CREATE macro.
12 (0C)	502 (1F6)	**Equate symbol**: IEFSSI_SWAP_ALREADY_ACTIVE **Meaning**: The SSVT that is to be made active (specified by the INTOKEN field) is already active. This is a SWAP request error. **Action**: None.
12 (0C)	600 (258)	**Equate symbol**: IEFSSI_EVENTRTN_NOT_FOUND **Meaning**: A usable copy of the event notification routine could not be found. This is an OPTIONS request error. For instance: • The module was not found. • The module was found, but was not APF-authorized. **Action**: Correct the event notification routine name or make sure it is present in either LINKLIB or LPALIB, and is APF-authorized.
12 (0C)	900 (384)	**Equate symbol**: IEFSSI_QUERY_STORAGE **Meaning**: Unable to obtain storage for an output of the QUERY request. This a QUERY request error. **Action**: Check the current use of the system storage to determine why storage was not available. Retry the request later in case storage has become available.
20 (14)	—	**Equate symbol**: IEFSSI_SYSTEM_ERROR **Meaning**: System error **Action**: Investigate the following possible causes: • Inability to obtain a system resource • Abnormal task termination Obtain the system dump, if any, and contact the IBM support center.
24 (18)	—	**Equate symbol**: IEFSSI_UNAVAILABLE **Meaning**: The IEFSSI macro has been invoked too early during system initialization. **Action**: Delay the invocation of the IEFSSI macro to a later point in the IPL.

Example 1

Submit a request to add subsystem FRED, call the initialization routine and route all initialization messages to the FREDCONS console

```
         IEFSSI REQUEST=ADD,SUBNAME=SNAME,INITRTN=INITPGM,             X
               INITPARM=IPARMS,INITPLEN=5,CONSNAME=ICONSOLE,           X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
⋮
SNAME    DC    CL4'FRED'
INITPGM  DC    CL8'FREDPGM '
IPARMS   DC    CL60'HELLO'
ICONSOLE DC    CL8'FREDCONS'
⋮
WORKAREA DSECT 0F
RETURN_CODE DS F
REASON_CODE DS F
WORKLEN  EQU   *-WORKAREA
```

Example 2

Activate subsystem FRED using the SSVT identified by the SSVTTOK token. Assume that the SSVTTOK token was returned by a previous invocation of the IEFSSVT REQUEST=CREATE macro.

```
         IEFSSI REQUEST=ACTIVATE,SUBNAME=SNAME,INTOKEN=SSVTTOK,        X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
⋮
```

Example 3

Inform the system that the subsystem responds to SETSSI commands and requires the services of the primary subsystem when starting. Also specify an event notification routine to get control whenever a subsystem event occurs.

```
         IEFSSI REQUEST=OPTIONS,SUBNAME=SNAME,COMMAND=YES,REQDSUB=PRI, X
              EVENTRTN=EVENTPGM,RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
⋮
EVENTPGM  DC     CL8'EVNTPGM '
```

Example 4

Deactivate a subsystem and return the token of the outgoing SSVT.

```
         IEFSSI REQUEST=DEACTIVATE,SUBNAME=SNAME,OUTTOKEN=SSVTTOK,     X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 5

Replace the current set of function routines being used by the subsystem with a new set of function routines. NEWTOK is a token previously returned by the IEFSSVT REQUEST=CREATE service. NEWTOK identifies the incoming SSVT.

```
         IEFSSI REQUEST=SWAP,SUBNAME=SNAME,INTOKEN=NEWTOK,             X
               OUTTOKEN=OLDTOK,                                        X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 6

Store the address of the FREDCB subsystem control block for later retrieval by the subsystem function routines.

```
         LA    5,FREDCB    Get address of subsystem control block
         ST    5,DATA1     Store address
         LA    4,DATA1     Get address of field containing pointer
```

```
         IEFSSI REQUEST=PUT,SUBNAME=SNAME,SUBDATA1=(4),                X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Note: When using the register notation (4), the register contains the address of the data to be stored, not the data itself. The data stored in this case is the address of the FREDCB control block.

Example 7

Retrieve subsystem-defined data that was previously stored using the IEFSSI REQUEST=PUT service and place the retrieved data at the location whose address is contained in register 5.

```
         IEFSSI REQUEST=GET,SUBNAME=SNAME,SUBDATA1=(5),                X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 8

Obtain subsystem information for all subsystems whose name begins with 'JES' and free the storage obtained by the SSI.

```
         IEFSSI REQUEST=QUERY,SUBNAME=SNAME,                           X
               WORKAREA=WAREA,                                         X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
⋮
         L      R5,WAREA
         USING  JQRY_HEADER,R5
         L      R0,JQRYLEN
         STORAGE RELEASE,LENGTH=(0),ADDR=(R5)
⋮
SNAME    DC    CL4'JES*'
WAREA    DS    A
         IEFJSQRY
```

Chapter 91. IEFSSVT — Create a subsystem vector table

Description

Use the IEFSSVT macro to:

- Create subsystem vector tables (SSVTs).
- Modify the subsystem response to function requests by:
  - Disabling existing function codes
  - Enabling new function codes
  - Exchange function routines for a supported function code

The IEFSSVT macro allows users to specify function routines by address or name rather than requiring the subsystem interface (SSI) to load the routines. This is useful if the subsystem wants to load its function routines into global storage, but does not want the routines to be deleted if the address space ends. In this case, the subsystem can perform a load-to-address, rather than a standard load, and pass the addresses to the IEFSSVT macro.

The requests for the macro are:

- IEFSSVT REQUEST=CREATE, which creates an SSVT. See in "REQUEST=CREATE parameter of IEFSSVT" on page 913 for the syntax of this request.
- IEFSSVT REQUEST=DISABLE, which disables supported function codes. See in "REQUEST=DISABLE parameter of IEFSSVT" on page 917 for the syntax of this request.
- IEFSSVT REQUEST=ENABLE, which enables additional function codes. See in "REQUEST=ENABLE parameter of IEFSSVT" on page 920 for the syntax of this request.
- IEFSSVT REQUEST=EXCHANGE, which replaces the function routine associated with a supported function code with another function routine. See in "REQUEST=EXCHANGE parameter of IEFSSVT" on page 923 for the syntax of this request.

For ease of use, the standard form of the macro is shown for each IEFSSVT request. The requests are described on the following pages along with the:

- Standard form syntax diagram
- Description of the parameters

The following information is described once at the beginning of the IEFSSVT macro description:

- Environment
- Programming requirements
- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:

- Abend codes
- Return and reason codes
- Examples

The REQUEST=CREATE, REQUEST=DISABLE, REQUEST=ENABLE, and REQUEST=EXCHANGE parameters are mutually exclusive. You can select only one.

For information about using dynamic subsystem services, see *z/OS MVS Using the Subsystem Interface*. This topic also includes information about related macros IEFSSVTI and IEFSSI.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • Any system key • PSW key mask (PKM) allowing key 0-7 • APF authorization
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24-bit or 31-bit
ASC mode:	Primary or Access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space

Programming requirements

Before invoking IEFSSVT, you must invoke IEFSSVTI to build a table of function routines and function codes as input to IEFSSVT.

Also:

- Include the CVT and IEFJESCT mapping macros in your program.
- Include the IEFJSRC mapping macro in your program. This macro defines the dynamic SSI return and reason codes.

Restrictions

The services that IEFSSVT provides are available only to dynamic subsystems, which are those subsystems that have been defined to the SSI in one of the following ways:

- Processing the keyword format of the IEFSSNxx parmlib member during IPL
- Issuing the IEFSSI macro
- Issuing the SETSSI system command.

Input register information

Before issuing the IEFSSVT macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system.

2-13
Unchanged

14-15
Used as a work register by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

REQUEST=CREATE parameter of IEFSSVT

The IEFSSVT macro with the CREATE parameter builds the SSVT. An SSVT built with the IEFSSVT REQUEST=CREATE is referred to as an SSI-managed vector table. Only SSI-managed SSVTs can take advantage of the dynamic SSI services. See *z/OS MVS Using the Subsystem Interface* for more information about dynamic SSI services.

Syntax for REQUEST=CREATE

The syntax of the IEFSSVT REQUEST=CREATE macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVT.
IEFSSVT	
ƀ	One or more blanks must follow IEFSSVT.

Syntax	Description
SUBNAME=subname	
,REQUEST=CREATE	
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,OUTTOKEN=*outtoken*	*outtoken*: RS-type address or address in register (2) - (12).
,SUBPOOL=*subpool*	*subpool:*. RS-type address or address in register (2) - (12).
,SUBPOOL=*241*	**Default**: SUBPOOL=241
,MAXENTRIES=*maxentries*	*maxentries*: RS-type address or address in register (2) - (12).
,LOADTOGLOBAL=*NO*	**Default**: LOADTOGLOBAL=NO
,LOADTOGLOBAL=*YES*	
,ERRFUNCT=*errfunct*	*errfunct*: RS-type address or address in register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable.
,COM=*com*	*com*: comment string
,COM=*NULL*	**Default**: COM=NULL
,MF=S	
,MF=(L,*list addr*	*list addr*: symbol.
,MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
,MF=(L,*list addr*,0D)	**Default:** 0D

Syntax	Description
,MF=(E,*list addr*)	*list addr*: RS-type address or address in register (1) - (12) of a storage area.
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters for REQUEST=CREATE

The parameters are explained as follows:

SUBNAME=subname
A required 4-character parameter that specifies the field (or an address in a register) containing the subsystem name. It must be the name of a subsystem that has been previously defined to the system using dynamic SSI services. See *z/OS MVS Using the Subsystem Interface* for more information about dynamic SSI services.

This field must be padded to the right with blanks or nulls if it is less than 4 characters long.

,REQUEST=CREATE
A parameter that specifies an SSVT is to be built and initialized.

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the function routine input table that associates supported SSVT function codes with function routines. Use the IEFSSVTI macro to build the table. Use this field to contain the name specified by the IEFSSVTI SSVTDATA parameter.

,OUTTOKEN=*outtoken*
A required 32-bit parameter that specifies the name (or an address in a register) of an output token used to identify the SSVT.

,SUBPOOL=*subpool*
,SUBPOOL=*241*
An optional byte parameter that specifies the name (or an address in a register) of an input field that contains the subpool number in which the SSVT is to be built. The subpool must represent common storage. The default is 241.

,MAXENTRIES=*maxentries*
A required halfword parameter that specifies the name (or an address in a register) of an input field that defines the maximum number of function routine entries that the SSVT can contain. The maximum number must be:

- Greater than or equal to 1 and less than or equal to 255
- Greater than or equal to the number of function routines defined in the input table created with the IEFSSVTI macro.
- The maximum number of function routine entries the calling subsystem requires for the SSVT.

Note: Consider that the value for MAXENTRIES should provide for additional function routines that can be defined through the SET or EXCHANGE parameters of the IEFSSVT service.

,LOADTOGLOBAL=*NO*
,LOADTOGLOBAL=*YES*
An optional parameter that specifies that the function routines are to be loaded to global storage. Use this parameter when the function routines:

- Are specified by name in the function routine input table that the IEFSSVTI macro created
- Do not reside in the link pack area.

This parameter is ignored if the input function routines are specified by address.

The meanings are:

NO
Indicates that a load-to-global is not necessary for the function routines. No is the default.

YES
Indicates that a load-to-global is necessary for the function routines. The SSI issues a LOAD for each named function routine with the following parameters:

- EOM=YES
- LSEARCH=NO
- GLOBAL=(YES,P)

,ERRFUNCT=*errfunct*
An optional 8-character parameter that specifies the name (or an address in a register) of an output field that receives the function routine name being processed when an error occurred. Check this output field if you get return code IEFSSVT_LOAD_ERROR (decimal 16) from the IEFSSVT macro.

PLISTVER=IMPLIED_VERSION
PLISTVER=MAX
PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=*NULL*
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=DISABLE parameter of IEFSSVT

The IEFSSVT macro with the DISABLE parameter disables supported function codes.

Syntax for REQUEST=DISABLE

The syntax of the IEFSSVT REQUEST=DISABLE macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVT.
IEFSSVT	
ƀ	One or more blanks must follow IEFSSVT.
SUBNAME=subname	
,REQUEST=DISABLE	

Syntax	Description
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,INTOKEN=*intoken*	*intoken*: RS-type address or address in register (2) - (12).
,INTOKEN=*NULL*	**Default**: INTOKEN=NULL
,PLISTVER=IMPLIED_VERSION	**Default**: IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable.
,COM=*com*	*com*: comment string
,COM=*NULL*	**Default**: COM=NULL
,MF=S	
,MF=(L,*list addr*	*list addr*: symbol.
,MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
,MF=(L,*list addr*,0D)	**Default:** 0D
,MF=(E,*list addr*)	*list addr*: RS-type address or address in register (1) - (12) of a storage area.
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters for REQUEST=DISABLE

The parameters are explained as follows:

SUBNAME=subname
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it less than 4 characters long.

,REQUEST=DISABLE
A parameter that specifies that you want to disable function codes.

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the function routine input table that identifies the function codes that you want to disable. The IEFSSVTI macro has built the table. This field contains the name specified by the IEFSSVTI SSVTDATA parameter.

For the disable request, the system uses only the function code information. The function routine names or addresses in the input table are ignored.

,INTOKEN=*intoken*
,INTOKEN=*NULL*
An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the target SSVT (OUTTOKEN from REQUEST=CREATE). The default is NULL.

PLISTVER=IMPLIED_VERSION
PLISTVER=MAX
PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=*NULL*
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=ENABLE parameter of IEFSSVT

The IEFSSVT macro with the ENABLE parameter activates new function codes or reactivates previously disabled function codes. You can enable new function codes only if the SSVT has available function routine slots to contain any new function routines. An ENABLE request may not need to specify new function routines, if the routine that supports a new code is already supporting a previously enabled code.

Syntax for REQUEST=ENABLE

The syntax of the IEFSSVT REQUEST=ENABLE macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVT.
IEFSSVT	
ƀ	One or more blanks must follow IEFSSVT.
SUBNAME=subname	

Syntax	Description
,REQUEST=ENABLE	
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,INTOKEN=*intoken*	*intoken*: RS-type address or address in register (2) - (12).
,INTOKEN=*NULL*	**Default**: INTOKEN=NULL
,LOADTOGLOBAL=*NO*	**Default**: LOADTOGLOBAL=NO
,LOADTOGLOBAL=*YES*	
,ERRFUNCT=*errfunct*	*errfunct*: RS-type address or address in register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable.
,COM=*com*	*com*: comment string
,COM=*NULL*	**Default**: COM=NULL
,MF=S	
,MF=(L,*list addr*	*list addr*: symbol.
,MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
,MF=(L,*list addr*,0D)	**Default:** 0D
,MF=(E,*list addr*)	*list addr*: RS-type address or address in register (1) - (12) of a storage area.
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters for REQUEST=ENABLE

The parameters are explained as follows:

SUBNAME=subname
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it less than 4 characters long.

,REQUEST=ENABLE
A parameter that specifies that you want to enable the function codes specified in the SSVTDATA parameter.

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the function routine input table that identifies the new function codes that the SSVT supports and their related function routines. Use the IEFSSVTI macro to build the table. This field contains the the name specified by the IEFSSVTI SSVTDATA parameter.

,INTOKEN=*intoken*
,INTOKEN=*NULL*
An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the target SSVT. The default is NULL.

,LOADTOGLOBAL=*NO*
,LOADTOGLOBAL=*YES*
An optional parameter that specifies that the function routines are to be loaded to global storage. Use this parameter when the function routines:

- Are specified by name in the function routine input table that the IEFSSVTI macro created
- Do not reside in the link pack area.

This parameter is ignored if the input function routines are specified by address.

The meanings are:

NO
Indicates that a load-to-global is not necessary for the function routines. This is the default.

YES
Indicates that a load-to-global is necessary for the function routines. The SSI issues a LOAD for the named function routine with the following parameters:

- EOM=YES
- LSEARCH=NO
- GLOBAL=(YES,P)

,ERRFUNCT=*errfunct*
An optional 8-character parameter that specifies the name (or an address in a register) of an output field that receives the function routine name being processed when an error occurred. Check this output field if you get return code IEFSSVT_LOAD_ERROR (decimal 22) from the IEFSSVT macro.

PLISTVER=IMPLIED_VERSION
PLISTVER=MAX
PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=*NULL*
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

REQUEST=EXCHANGE parameter of IEFSSVT

The IEFSSVT macro with the EXCHANGE parameter exchanges the function routine that supports a function code with a different function routine.

Syntax for REQUEST=EXCHANGE

The syntax of the IEFSSVT REQUEST=EXCHANGE macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVT.
IEFSSVT	
ƀ	One or more blanks must follow IEFSSVT.
SUBNAME=subname	
,REQUEST=EXCHANGE	
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,INTOKEN=*intoken*	*intoken*: RS-type address or address in register (2) - (12).
,INTOKEN=*NULL*	**Default**: INTOKEN=NULL
,LOADTOGLOBAL=*NO*	**Default**: LOADTOGLOBAL=NO
,LOADTOGLOBAL=*YES*	
,ERRFUNCT=*errfunct*	*errfunct*: RS-type address or address in register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 1
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode*	*rsncode*: RS-type address or address in register (2) - (12) of fullword output variable.
,COM=*com*	*com*: comment string

Syntax	Description
,COM=*NULL*	**Default**: COM=NULL
,MF=S	
,MF=(L,*list addr*	*list addr*: symbol.
,MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
,MF=(L,*list addr*,0D)	**Default:** 0D
,MF=(E,*list addr*)	*list addr*: RS-type address or address in register (1) - (12) of a storage area.
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters for REQUEST=EXCHANGE

The parameters are explained as follows:

SUBNAME=subname
A required parameter that specifies the field (or an address in a register) containing the 4-character subsystem name. It must be the name of a subsystem that has been previously defined to the system using SSI services.

This field must be padded to the right with blanks or nulls if it less than 4 characters long.

,REQUEST=EXCHANGE
A parameter that specifies that you want to exchange existing function routines with new function routines.

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the function routine input table that identifies the new function codes affected by the exchange and the new function routines that support them. Use the IEFSSVTI macro to build the table. This field contains the the name specified by the IEFSSVTI SSVTDATA parameter.

,INTOKEN=*intoken*
,INTOKEN=*NULL*
An optional 32-bit parameter that specifies the name (or an address in a register) of an input token that represents the target SSVT. The default is NULL.

,LOADTOGLOBAL=*NO*
,LOADTOGLOBAL=*YES*
An optional parameter that specifies that the function routines are to be loaded to global storage. Use this parameter when the function routines:

- Are specified by name in the function routine input table that the IEFSSVTI macro created
- Do not reside in the link pack area.

This parameter is ignored if the input function routines are specified by address.

NO
Indicates that a load-to-global is not necessary for the function routines. This is the default.

YES
Indicates that a load-to-global is necessary for the function routines. The SSI issues a LOAD for the named function routine with the following parameters:

- EOM=YES

- LSEARCH=NO
- GLOBAL=(YES,P)

,ERRFUNCT=*errfunct*
An optional 8-character parameter that specifies the name (or an address in a register) of an output field that receives the function routine name being processed when an error occurred. Check this output field if you get return code IEFSSVT_LOAD_ERROR (decimal 22) from the IEFSSVT macro.

PLISTVER=IMPLIED_VERSION
PLISTVER=MAX
PLISTVER=*plistver*
Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

MAX
The largest size parameter list currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
The currently available parameters.

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,RETCODE=*retcode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the return code. The return code is copied from general purpose register 15.

,RSNCODE=*rsncode*
An optional 4-byte parameter that specifies a name (or register) of an output field where the system places the reason code. The reason code is copied from general purpose register 0.

,COM=*com*
,COM=*NULL*
An optional parameter that specifies the character input that appears in the block comment before the macro invocation. Use it to make comments about the macro invocation. The comment must be enclosed in quotation marks if it contains any lower case characters. The default is NULL.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Use MF=S to specify the standard form of the IEFSSVT macro, which builds an in-line parameter list and generates the macro invocation to transfer control to the service.

Use MF=L to specify the list form of the IEFSSVT macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage

that the execute form uses to store the parameters. No other parameters may be coded with the list form of the macro.

Use the MF=E together with the list form of the macro for applications that require reentrant code. The execute form of the IEFSSVT macro stores the parameters into the storage area defined by the list form and generates the macro invocation to transfer control to the service.

,list addr
A required parameter that specifies the name of a storage area for the parameter list.

,attr
An optional 1-to 60-character input string that contains any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
An optional parameter that specifies that the system checks for required parameters and supply defaults for omitted optional parameters. This is the default parameter.

ABEND codes

An invocation of the IEFSSVT macro may result in an abend code X'8C5'. See *z/OS MVS System Codes* for an explanation of this abend code.

Return and reason codes

When the IEFSSVT macro returns control to your program, GPR 15 (and *retcode*, if you coded RETCODE) contains a return code. When the value in GPR 15 is not 0, GPR 0 (and *rsncode* if you coded RSNCODE) contains the reason code.

The IEFJSRC mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each return code are:

Return Code Decimal (Hex)	Equate Symbol for Return Code
00 (00)	IEFSSVT_SUCCESS
04 (04)	IEFSSVT_WARNING
08 (08)	IEFSSVT_INVALID_PARAMETERS
12 (0C)	IEFSSVT_REQUEST_FAIL
16 (010)	IEFSSVT_LOAD_ERROR
20 (014)	IEFSSVT_SYSTEM_ERROR
24 (018)	IEFSSVT_UNAVAILABLE

The following table contains return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 86. Return and Reason Codes for the IEFSSVT Macro

Return Code Decimal (hex)	Reason Code Decimal (hex)	Equate Symbol for Reason Code Meaning and Action
00 (00)	00 (00)	**Equate Symbol**: IEFSSVT_FUNCTIONS_COMPLETE **Meaning**: The request completed successfully. The result depends on the request: • CREATE — An SSVT is created. • DISABLE — Supported function codes are disabled. • ENABLE — Additional function codes are enabled. • EXCHANGE — New function routines replace supported function routines. **Action**: None.
04 (04)	00 (00)	**Equate Symbol**: None
08 (08)	00 (00)	**Equate Symbol**: IEFSSVT_SUBSYSTEM_UNKNOWN **Meaning**: The subsystem was not defined to the SSI. **Action**: None.
08 (08)	04 (004)	**Equate Symbol**: IEFSSVT_NON_DYNAMIC **Meaning**: The subsystem is not a dynamic subsystem. **Action**: None.
08 (08)	08 (008)	**Equate Symbol**: IEFSSVT_BAD_VT_TOKEN **Meaning**: The SSVT token does not correspond to a valid SSVT. **Action**: None.
08 (08)	12 (00C)	**Equate Symbol**: IEFSSVT_INVALID_NAME **Meaning**: The subsystem name or the routine name contains characters that are not valid. **Action**: None.
08 (08)	16 (010)	**Equate Symbol**: IEFSSVT_INVALID_FUNCTION_CODE **Meaning**: The function code is outside the valid range. **Action**: Correct the function code.
08 08	20 (014)	**Equate Symbol**: IEFSSVT_DUPLICATE_FUNCTION_CODE **Meaning**: The function code appears more than once in the function routine input table. **Action**: Delete the duplicate specification.
08 08	24 (018)	**Equate Symbol**: IEFSSVT_INVALID_FROUTINE **Meaning**: The function routine name or address is null. **Action**: If working with a function routine input table in dynamic storage, use the IEFSSVTI SET function to identify the function routine by name or address. If using a static table, the function routine must be identified by name using the IEFSSVTI ENTRY function.
08 08	28 (01C)	**Equate Symbol**: IEFSSVT_NO_FCODES **Meaning**: The function routine entry in the function routine input table does not specify any function codes. **Action**: Specify the function codes with either the IEFSSVTI ENTRY or SET function.
12 (0C)	00 (00)	**Equate Symbol**: None

Table 86. Return and Reason Codes for the IEFSSVT Macro (continued)

Return Code Decimal (hex)	Reason Code Decimal (hex)	Equate Symbol for Reason Code Meaning and Action
12 (0C)	100 (064)	**Equate Symbol**: IEFSSVT_MAX_VECTOR_TABLES **Meaning**: The maximum number of SSVTs already exists for the subsystem. This is a CREATE request error. **Action**: Additional vector tables cannot be created during the current IPL. Use the IEFSSVT ENABLE and DISABLE services to modify the response of an existing vector table.
12 (0C)	101 (065)	**Equate Symbol**: IEFSSVT_STORAGE **Meaning**: Unable to obtain storage for an SSVT. This is a CREATE request error. **Action**: Review the use of system storage to determine why sufficient storage was not available. Try the request again later.
12 (0C)	102 (066)	**Equate Symbol**: IEFSSVT_MAXENTRIES_TOO_SMALL **Meaning**: The MAXENTRIES value is less than the number of function routines in the function routine input table. This is a CREATE request error. **Action**: Correct the MAXENTRIES value. It must be at least as large as the number of function routines represented in the function routine input table.
12 (0C)	103 (067)	**Equate Symbol**: IEFSSVT_MAXENTRIES_TOO_BIG **Meaning**: The MAXENTRIES value is greater than the maximum value (255). This is a CREATE request error. **Action**: None.
12 (0C)	200 (0C8)	**Equate Symbol**: IEFSSVT_ENABLE_NO_ELIGIBLE_VT **Meaning**: The SSVT is not specified and a valid default is not available. This is an ENABLE request error. **Action**: Use the IEFSSVT CREATE function to create an SSI-managed vector table.
12 (0C)	201 (0C9)	**Equate Symbol**: IEFSSVT_ENABLE_MAX_ROUTINES **Meaning**: The SSVT does not have available space for new function routines. This is an ENABLE request error. **Action**: If the subsystem has only one SSI-managed vector table, use the IEFSSVT CREATE function to create a second larger one, which responds to all required function codes. If the subsystem already has two vector tables, the problem cannot be corrected without re-IPLing the system, unless some function codes can be disabled to make room for the new required function routines.
12 (0C)	202 (0CA)	**Equate Symbol**: IEFSSVT_FUNCTION_ALREADY_ENABLED **Meaning**: The subsystem already responds to one of the function codes for which this request was submitted. This an ENABLE request error. **Action**: If you want to change the routine that supports the function , use the IEFSSVT EXCHANGE function.
12 (0C)	300 (12C)	**Equate Symbol**: IEFSSVT_DISABLE_NO_ELIGIBLE_VT **Meaning**: The SSVT is not specified and a valid default is not available. This an DISABLE request error. **Action**: Create an SSI-managed SSVT using the IEFSSVT CREATE function.
12 (0C)	500 (1F4)	**Equate Symbol**: IEFSSVT_EXCHANGE_NO_ELIGIBLE_VT **Meaning**: The SSVT is not specified and a valid default is not available. This an EXCHANGE request error. **Action**: Create an SSI-managed SSVT using the IEFSSVT CREATE function.

Table 86. Return and Reason Codes for the IEFSSVT Macro (continued)		
Return Code Decimal (hex)	**Reason Code Decimal (hex)**	**Equate Symbol for Reason Code Meaning and Action**
12 (0C)	501 (1F5)	**Equate Symbol**: IEFSSVT_EXCHANGE_MAX_ROUTINES **Meaning**: The SSVT does not have available space for new function routines. This an EXCHANGE request error. **Action**: If the subsystem has only one SSI-managed vector table, use the IEFSSVT CREATE function to create a second larger one, which responds to all required function codes. If the subsystem already has two vector tables, the problem cannot be corrected without re-IPLing the system, unless some function codes can be disabled to make room for the new required function routines.

Examples

For the following examples, assume that the function routine input tables have already been built using the IEFSSVTI macro.

Example 1

Create an SSVT, reserving space for 5 function routines. The function routines reside in LPA, so the LOADTOGLOBAL parameter is allowed to default to NO.

```
         IEFSSVT REQUEST=CREATE,SUBNAME=SNAME,SSVTDATA=FROUTINE_TABLE, X
               MAXENTRIES=5,OUTTOKEN=NEWTOKEN,ERRFUNCT=BADNAME,        X
               RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
⋮
SNAME    DC    CL4'FRED'
⋮
WORKAREA DSECT 0F
NEWTOKEN DS    CL4
BADNAME  DS    CL8
RETURN_CODE DS F
REASON_CODE DS F
WORKLEN  EQU   *-WORKAREA
```

Example 2

Enable an additional function code in the SSVT that was created in example 1. The function routine input table ENABLE_TABLE describes only the new function code and its associated function routine.

```
IEFSSVT REQUEST=ENABLE,SUBNAME=SNAME,SSVTDATA=ENABLE_TABLE,   X
      INTOKEN=NEWTOKEN,ERRFUNCT=BADNAME,                      X
      RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 3

Disable one of the function codes currently supported by the SSVT created in example 1. The function routine input table DISABLE_TABLE describes only the function code to be disabled. It does not have to provide function routine information.

```
IEFSSVT REQUEST=DISABLE,SUBNAME=SNAME,SSVTDATA=DISABLE_TABLE, X
      INTOKEN=NEWTOKEN,                                       X
      RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Example 4

Change the function routine that responds to one of the function codes supported by the SSVT in example 1. The function routine input table EXCHANGE_TABLE identifies the function code and the new function routine.

```
IEFSSVT REQUEST=EXCHANGE,SUBNAME=SNAME,
      SSVTDATA=EXCHANGE_TABLE,                                  X
      INTOKEN=NEWTOKEN,ERRFUNCT=BADNAME,                        X
      RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

Chapter 92. IEFSSVTI — Associate function routines with function codes

Description

Use the IEFSSVTI macro to create a table that relates function routines to the function codes they support. This information is passed to the IEFSSVT macro as input when creating or modifying an SSVT.

The IEFSSVTI macro:

- Creates a static function routine input table
- Reserves dynamic storage for a function routine input table
- Copies a static table to dynamic storage
- Modifies a function routine input table in dynamic storage

A **static** function routine input table is a table that does not change at program run time and is used when all the information, such as the function routine names, are known at compile time.

The IEFSSVTI macro does not attempt to verify that its caller is a dynamic subsystem. IEFSSVTI can be used only with IEFSSVT.

Each function routine defined by the SET or ENTRY parameters of the IEFSSVTI macro occupies a separate entry in the SSVT. The SSVT size is limited to the number of entries specified by the MAXENTRIES parameter at the time when the SSVT is created (IEFSSVT CREATE). As a result, IBM suggests economizing the use of the slots by identifying only unique function routines for each SET or ENTRY request of IEFSSVTI. In this way, if a function routine is common to several function codes, a single call is made to IEFSSVTI to relate all of the function codes to the function routine, rather than calling IEFSSVTI many times, relating the same function routine to many function codes. For examples of relating multiple function codes to a single function routine, see “Example 1” on page 936, which identifies the LISTEN function routine related to two function codes, or see “Example 2” on page 936, which identifies the VERSION function routine, related to two function codes.

Note: The IEFSSVTI macro expands in-line and therefore does not impose restrictions on authorization, dispatch mode, cross-memory mode, locks or control parameters. The program using the table created by the IEFSSVTI macro is subject to the restrictions of the IEFSSVT macro.

The types for the macro are:

- IEFSSVTI TYPE=LIST, which creates a DSECT that maps the format of the function routine input table. See in “TYPE=LIST parameter of IEFSSVTI” on page 937 for the syntax of this request.
- IEFSSVTI TYPE=INITIAL, which begins the definition of a static function routine input table. See in “TYPE=INITIAL parameter of IEFSSVTI” on page 937 for the syntax of this request.
- IEFSSVTI TYPE=ENTRY, which defines a function routine entry in a static input table. See in “TYPE=ENTRY parameter of IEFSSVTI” on page 938 for the syntax of this request.
- IEFSSVTI TYPE=FINAL, which ends the definition of a static function routine input table. See in “TYPE=FINAL parameter of IEFSSVTI” on page 940 for the syntax of this request.
- IEFSSVTI TYPE=SET, which modifies a function routine entry in an existing input table. See in “TYPE=SET parameter of IEFSSVTI” on page 941 for the syntax of this request.
- IEFSSVTI TYPE=RESERVE, which reserves storage for a function routine input table. See in “TYPE=RESERVE parameter of IEFSSVTI” on page 943 for the syntax of this request.
- IEFSSVTI TYPE=COPY, which copies a function routine input table. See in “TYPE=COPY parameter of IEFSSVTI” on page 944 for the syntax of this request.

For ease of use, the standard form of the macro is shown for each IEFSSVTI type. The types are described on the following pages along with the:

- Standard form syntax diagram
- Description of the parameters

The following information is described once at the beginning of the IEFSSVTI macro description:

- Environment
- Programming requirements
- Restrictions
- Input register information
- Output register information
- Performance implications

Following the descriptions of the standard forms of all requests are:

- Abend codes
- Return and reason codes
- Examples

The TYPE=LIST, TYPE=INITIAL, TYPE=ENTRY, TYPE=FINAL, TYPE=SET, TYPE=RESERVE, and TYPE=COPY parameters are mutually exclusive. You can select only one.

For information about using dynamic subsystem services, see *z/OS MVS Using the Subsystem Interface*. This topic also includes information about the related macro IEFSSI.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24-bit or 31-bit
ASC mode:	Primary or Access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space

Programming requirements

If the subsystem function routines are identified by address using the FUNCADDR parameter, the invoking program must load the function routines by using the LOAD macro, or obtain their addresses before invoking this macro.

Restrictions

The input table this macro creates:

- Can only be used with the IEFSSVT macro
- Cannot be used by the IEFJSVEC service to create SSVTs

The register form can be used to specify macro parameter variables only in TYPE=SET invocations.

Input register information

Before issuing the IEFSSVTI macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

ABEND codes

None. The IEFSSVT macro indicates whether the IEFSSVTI macro processing was successful, because you must use the IEFSSVTI macro with the IEFSSVT macro when creating an SSVT.

Return and reason codes

None.

Examples

The following notes apply to the supplied examples:

Note:

1. A set of IEFSSVTI macro invocations, beginning with TYPE=INITIAL and ending with TYPE=FINAL must contain some invocations specifying FUNCNAME and some specifying FUNCADDR. The input table does not have to identify all function routines the same way.
2. Usage scenarios

   When you know all of the information at compile time, you can create a static function routine input table using: TYPE=INITIAL, TYPE=ENTRY, and TYPE=FINAL.

   When you do not know all of the information at compile time, you can create a dynamic function routine input table as follows:

- For reentrant programs: create a static function routine input table, reserve storage for a dynamic table of the same size using TYPE=RESERVE, copy the static table to the dynamic table using TYPE=COPY, and modify the dynamic table using TYPE=SET.
- For non-reentrant programs: create a static table and modify it using TYPE=SET, or copy a static table to dynamic storage and modify the dynamic table using TYPE=SET.

Example 1

Build a static function routine input table, specifying the function routines by name.

```
IEFSSVTI TYPE=INITIAL,SSVTDATA=MY_STAT_TABLE,                 +
      TABLEN=STAT_TABLE_LENGTH
IEFSSVTI TYPE=ENTRY,FUNCNAME=LISTEN,NUMFCODES=2,              +
      FCODES=(SSOBWTO,SSOBWTL)
IEFSSVTI TYPE=ENTRY,FUNCNAME=VERSION,NUMFCODES=1,             +
      FCODES=SSOBSSVI
IEFSSVTI TYPE=FINAL
```

Example 2

Build a dynamic function routine input table, specifying function routines by address. A static function routine input table is defined as a template and copied to dynamic storage reserved by an IEFSSVTI TYPE=RESERVE invocation. The dynamic function routine input table is completed using TYPE=SET invocation.

```
INITRTN  CSECT
⋮
*********************************************************************
*   LOAD the function routines, store the entry point addresses,
*   and DELETE.
*********************************************************************
         LOAD EP=LISTEN
         ST    R0,LISTEN_ADDR
         DELETE EP=LISTEN
*
         LOAD EP=VERSION
         ST    R0,VERSION_ADDR
         DELETE EP=VERSION
*
*********************************************************************
*   Copy the static table to dynamic storage
*********************************************************************
         IEFSSVTI TYPE=COPY,SSVTDATA=MY_DYN_TABLE,                    +
               SOURCE=MY_TABLE
*
*********************************************************************
*  Set the function routine address information in the input
*  table.  Override the function codes supported for the first
*  entry, so that only SSOBWTO is supported and not SSOBWTL.
*********************************************************************
         IEFSSVTI TYPE=SET,SSVTDATA=MY_DYN_TABLE,SOURCE=MY_TABLE,      +
               ENTRYDATA=1,FUNCADDR=LISTEN_ADDR,FCODES=SSOBWTO
*
         IEFSSVTI TYPE=SET,SSVTDATA=MY_DYN_TABLE,SOURCE=MY_TABLE,      +
               ENTRYDATA=2,FUNCADDR=VERSION_ADDR
⋮
*********************************************************************
* Working storage
*********************************************************************
WORKAREA DSECT 0F
LISTEN_ADDR   DS  A                Address of listen function         +
                                      routine
VERSION_ADDR  DS  A                Address of version info function   +
                                      routine
*********************************************************************
* Reserve storage for dynamic function routine input table
*********************************************************************
         IEFSSVTI TYPE=RESERVE,SSVTDATA=MY_DYN_TABLE,                 +
               TABLEN=STAT_TABLE_LENGTH
*
WORKLEN  EQU   *-WORKAREA          Length of work area
*
         IEFSSVTI TYPE=LIST        Generate table mappings
*
```

```
INITRTN  CSECT
*
*********************************************************************
*  Create static function routine input table
*********************************************************************
         IEFSSVTI TYPE=INITIAL,SSVTDATA=MY_STAT_TABLE,                +
               TABLEN=STAT_TABLE_LENGTH
         IEFSSVTI TYPE=ENTRY,NUMFCODES=2,FCODES=(SSOBWTO,SSOBWTL)
         IEFSSVTI TYPE=ENTRY,NUMFCODES=1,FCODES=SSOBSSVI
         IEFSSVTI TYPE=FINAL
*
         END   INITRTN
```

TYPE=LIST parameter of IEFSSVTI

The IEFSSVTI macro with the LIST parameter produces a DSECT that maps the format of the function routine input table.

Syntax

The syntax of the IEFSSVTI macro with TYPE=LIST is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
ƀ	One or more blanks must follow IEFSSVTI.
TYPE=LIST	

Parameters

The parameters are explained as follows:

TYPE=LIST
A parameter that defines a DSECT that maps the format of the function routine input table. A TYPE=LIST request is required if a TYPE=SET request or TYPE=COPY request is used in the calling program.

TYPE=INITIAL parameter of IEFSSVTI

The IEFSSVTI macro with the INITIAL parameter begins the definition of a static function routine input table.

Syntax

The syntax of the IEFSSVTI macro with TYPE=INITIAL is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
ƀ	One or more blanks must follow IEFSSVTI.
TYPE=INITIAL	
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,TABLEN=*tablen*	*ssvtdata*: RS-type address

Parameters

The parameters are explained as follows:

TYPE=INITIAL
A parameter that begins a static function routine input table build request. This TYPE=INITIAL request is the first request required to build the static function routine input table. The order in which you invoke requests to build the static function routine input table follows:

- TYPE=INITIAL
- TYPE=ENTRY
- TYPE=FINAL

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the function routine input table that you are building that relates supported SSVT function codes with function routines. Use this name when referencing the function routine input table and also on the SSVTDATA parameter of the IEFSSVT macro.

,TABLEN=*tablen*
A required parameter that specifies the name of a constant that the IEFSSVTI macro generates to define the length of the storage required by the function routine input table.

Use this parameter with a TYPE=RESERVE request to reserve dynamic storage when copying the function routine input table for TYPE=SET request modifications.

TYPE=ENTRY parameter of IEFSSVTI

The IEFSSVTI macro with the ENTRY parameter defines a function routine entry in a static input table.

Syntax

The syntax of the IEFSSVTI macro with TYPE=ENTRY is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
ƀ	One or more blanks must follow IEFSSVTI.
TYPE=ENTRY	
,FUNCNAME=*funcname*	*funcname*: RS-type address or address in register (2) - (12).
,FUNCADDR=*funcaddr*	*funcaddr*: RS-type address or address in register (2) - (12). **Default**: none
,NUMFCODES=*numfcodes*	*numfcodes*: RS-type address or address in register (2) - (12).
,FCODES=*fcodes(,fcodes...)*	*fcodes*: RS-type address or address in register (2) - (12).
,FCODES=*0*	**Default**: FCODES=0

Parameters

The parameters are explained as follows:

TYPE=ENTRY
A parameter that defines a function routine entry in a static input table. A static input table must contain at least one TYPE=ENTRY invocation. You must specify a TYPE=INITIAL request prior to specifying this TYPE=ENTRY request.

,FUNCNAME=*funcname*
,FUNCADDR=*funcaddr*
An optional set of parameters. You can specify only one of the following:

FUNCNAME=*funcname*
The function routine name. This name can be no more than 8 characters long, beginning with an alphabetic character or national (#, @, or $) character. The remaining characters can be alphabetic, national or numeric.

This field must be left-justified and padded to the right with blanks.

For a TYPE=ENTRY request, if you omit FUNCNAME, you must provide the function routine information about a subsequent TYPE=SET request.

FUNCADDR=*funcaddr*
A field that contains the address of the function routine. Specifying FUNCADDR on a TYPE=ENTRY request, reserves storage in the function routine input table for the function routine address.

You must provide the actual address in a subsequent TYPE=SET request.

When you specify FUNCADDR on a TYPE=SET request, you can use the high-order bit to specify the function routine AMODE. Setting this bit indicates that the routine receives control in AMODE 31. Clearing this bit indicates that the routine receives control in AMODE 24. You can also use the FUNCAMODE key to indicate the AMODE of a function routine.

,NUMFCODES=*numfcodes*
A required 2-byte parameter that defines the number of function codes supported by the associated function routine. This input field contains a decimal value that must be in the range of 1 to 255.

Use the NUMFCODES parameter to reserve storage for function code information in a static function routine input table entry. You must specify a number greater or equal to the number of function codes specified with the FCODES parameter.

If you do not know the actual number of function codes associated with the routine, specify the maximum number of function codes you expect, to reserve enough storage. In this case, the FCODES parameter of a subsequent TYPE=SET request provides the actual function code information.

,FCODES=*fcodes(,fcodes...)*
,FCODES=*0*
An optional parameter that specifies the function codes supported by the associated function routine. This input field may contain either a value or a list of values that must be in the range of 1 to 255. The values do not have to be numbers, they can also be assembler equates.

The same function code value cannot appear more than once within a set of IEFSSVTI invocations representing a function routine input table.

For a TYPE=ENTRY request, if you do not specify the FCODES parameter, you must provide the supported function codes on a subsequent TYPE=SET request. The default is 0.

TYPE=FINAL parameter of IEFSSVTI

The IEFSSVTI macro with the FINAL parameter ends the definition of a static function routine input table.

Syntax

The syntax of the IEFSSVTI macro with TYPE=FINAL is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
ƀ	One or more blanks must follow IEFSSVTI.
TYPE=FINAL	

Parameters

The parameters are explained as follows:

TYPE=FINAL
A parameter that ends a static function routine input table build request. This TYPE=FINAL request is the last request required to build the static function routine input table.

TYPE=SET parameter of IEFSSVTI

The IEFSSVTI macro with the SET parameter modifies a function routine entry in an existing input table.

Syntax

The syntax of the IEFSSVTI macro with TYPE=SET is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
ƀ	One or more blanks must follow IEFSSVTI.
TYPE=SET	
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,SOURCE=*source*	*source*: RS-type address or address in register (2) - (12).
,ENTRYDATA=*entrydata*	*entrydata:* RS-type address or address in register (2) - (12).
,FUNCNAME=*funcname*	*funcname*: RS-type address or address in register (2) - (12).
,FUNCADDR=*funcaddr*	*funcaddr*: RS-type address or address in register (2) - (12). **Default**: none
,FUNCAMODE=*HOB*	**Default**: FUNCAMODE=HOB **Note:** FUNCAMODE is valid with FUNCADDR.
,FUNCAMODE=*31*	
,FUNCAMODE=*24*	
,FCODES=*fcodes(,fcodes...)*	*fcodes*: RS-type address or address in register (2) - (12).
,FCODES=*0*	**Default**: FCODES=0

Syntax	Description

Parameters

The parameters are explained as follows:

TYPE=SET
A parameter that modifies a function routine entry in an existing input table. You can use TYPE=SET to modify either a table in dynamic storage, or a static table in non-reentrant programs.

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the function routine input table to be modified. The name must match the name of a table specified on the SSVTDATA parameter of the TYPE=INITIAL or TYPE=RESERVE IEFSSVTI macro invocation.

,SOURCE=*source*
A required parameter that specifies the name of the original function routine input table from which the table to be modified was copied.

The name must match the name of a function routine input table that you specified in the SSVTDATA parameter on a TYPE=INITIAL invocation or a TYPE=RESERVE invocation. This information is used with the ENTRYDATA parameter to calculate the offset of the function routine input table you want to modify.

The SOURCE parameter and SSVTDATA parameter can refer to the same function routine input table. For example, you may want to modify a static function routine input table that was created by a set of TYPE=INITIAL and TYPE=FINAL invocations, which can be done only in non-reentrant modules.

,ENTRYDATA=*entrydata*
A required 4-byte parameter that specifies the name (or address) of the index of the function routine input table entry that you want to modify. This input field may be either a constant, an assembler equate, or decimal value.

If the value of the ENTRYDATA parameter is greater than the number of function routines in the source table, the target function routine input table does not change. If this is the case, you will not receive a compile-time warning message, because this situation is determined at run time.

,FUNCNAME=*funcname*
,FUNCADDR=*funcaddr*
An optional set of parameters. You can specify only one of the following:

FUNCNAME=*funcname*
The function routine name. This name can be no more than 8 characters long, beginning with an alphabetic character or national (#, @, or, $) character. The remaining characters can be alphabetic, national, or numeric.

This field must be left-justified and padded to the right with blanks.

For a TYPE=ENTRY request, if you omit FUNCNAME, you must provide the function routine information about this request.

FUNCADDR=*funcaddr*
The pointer input that contains the address of the function routine. Specifying FUNCADDR on a TYPE=ENTRY invocation, reserves storage in the function routine input table for the function routine address.

If you specify FUNCADDR on this invocation, you can use the high-order bit to specify the function routine AMODE. Setting this bit indicates that the routine receives control in AMODE 31. Clearing this bit indicates that the routine receives control in AMODE 24. You can also use the FUNCAMODE key to indicate the AMODE of a function routine.

,FUNCAMODE=*HOB*
,FUNCAMODE=*31*
,FUNCAMODE=*24*

An optional input parameter that specifies the AMODE of a function routine identified by the address. The SSI uses this information to determine the AMODE in which the function routine receives control.

If you do not specify the FUNCAMODE parameter, the high-order bit of the address specified with the FUNCADDR parameter indicates the AMODE of the function routine. If the high-order bit is on, the function routine is treated as AMODE 31.

FUNCAMODE=HOB specifies that the high-order bit of a function routine address indicates the AMODE in which the function routine receives control. HOB is the default.

FUNCAMODE=31 specifies that the function routine receives control in AMODE 31.

FUNCAMODE=24 specifies that the function routine receives control in AMODE 24.

,FCODES=*fcodes(,fcodes...)*
,FCODES=*0*

An optional byte parameter that specifies the function codes supported by the associated function routine. This input field contains a decimal value that must be in the range of 1 to 255.

The same function code value cannot appear more than once within a set of IEFSSVTI invocations representing a function routine input table.

You must specify the FCODES parameter if the function code information was not provided on the TYPE=ENTRY invocation that corresponds to the entry being modified.

Function codes that you specify with the TYPE=SET invocation replace any function codes specified on the original TYPE=ENTRY invocation. If you do not specify the FCODES parameter the function code information in the entry being modified is unchanged.

If you specify more function codes with the FCODES parameter than the maximum number of function codes for which room was reserved in the table entry being modified, the function code information in the target entry is unchanged. The IEFSSVTI macro does not provide a warning.

The default is 0.

TYPE=RESERVE parameter of IEFSSVTI

The IEFSSVTI macro with the RESERVE parameter reserves dynamic storage for a function routine input table.

Syntax

The syntax of the IEFSSVTI macro with TYPE=RESERVE is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
ƀ	One or more blanks must follow IEFSSVTI.

Syntax	Description
TYPE=RESERVE	
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,TABLEN=*tablen*	*tablen*: RS-type address
,MAXFCODES=*maxfcodes*	*maxfcodes*: RS-type address or address in register (2) - (12).

Parameters

The parameters are explained as follows:

TYPE=RESERVE
A parameter that reserves the amount of dynamic storage required to contain a copy of a static function routine input table.

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the function routine input table that relates supported SSVT function codes with function routines. Use this name when referencing the function routine input table and also on the SSVTDATA parameter of the IEFSSVT macro.

,TABLEN=*tablen*
,MAXFCODES=*maxfcodes*
You must specify one of the following parameters:

TABLEN=*tablen*
A parameter that specifies the name of a constant, which contains the length of storage required by the function routine input table. This should be the name of a constant specified by the TABLEN parameter on a previous TYPE=INITIAL invocation. Use this parameter when reserving storage for a dynamic function routine input table that is to be copied from another table.

MAXFCODES=*maxfcodes*
A parameter that specifies the maximum number of function codes that is supported by the entire function routine input table. Use this key to reserve space for a dynamic input table when the specific function routines and function codes that are supported are not known at compile time.

TYPE=COPY parameter of IEFSSVTI

The IEFSSVTI macro with the COPY parameter copies a static function routine input table to dynamic storage.

Syntax

The syntax of the IEFSSVTI macro with TYPE=COPY is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSVTI.

Syntax	Description
IEFSSVTI	
ƀ	One or more blanks must follow IEFSSVTI.
TYPE=COPY	
,SSVTDATA=*ssvtdata*	*ssvtdata*: RS-type address
,SOURCE=*source*	*source*: RS-type address or address in register (2) - (12).

Parameters

The parameters are explained as follows:

TYPE=COPY
A parameter copies a static function routine input table to dynamic storage.

The TYPE=COPY invocation expands inline to copy the table identified by the SOURCE parameter to the table identified by the SSVTDATA parameter. The source table contains the information for the macro to calculate the length that needs to be moved.

,SSVTDATA=*ssvtdata*
A required parameter that specifies the name of the target function routine input table (the destination for the copy). This name must match the name of a table specified by the SSVTDATA parameter on a TYPE=RESERVE invocation.

,SOURCE=*source*
A required parameter that specifies the name of the table to be copied. The name must match the name of a table that was specified on the SSVTDATA parameter on an TYPE=INITIAL invocation.

Chapter 93. IFAQUERY — SMF configuration query service

Description

The IFAQUERY service provides SMF configuration information to its caller. The IFAQUERY service currently performs the following function:

- RETRIEVE STATUS - Return the status of SMF recording. Information about SMF LOGSTREAMs is returned, including the SMF record types being written to the log stream. When SMF is not recording, a non-zero return code is returned.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and system PSW key.
Dispatchable unit mode:	Task mode
Cross Memory Mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

None.

Input register information

There are no input register requirements for issuing the IFAQUERY macro.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The standard form of the IFAQUERY macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IFAQUERY.
IFAQUERY	
ƀ	One or more blanks must follow IFAQUERY.
[,*xlabel*]	An optional symbol, starting in column 1, that is the name on the IFAQUERY macro invocation. DEFAULT: No name
RETRIEVE	
,STATUS	
,OUTAREA=*outarea*	*xoutarea*: RS-type address or register (2) - (12).
,OUTLEN=*outlen*	*outlen*: RS-type address or register (2) - (12).
[,DETAILS=LOGSTREAM]	**Default**: ALL

Syntax	Description
[,RETCODE=*retcode*]	*retcode*: RS-type address or register (2) - (12).
[,RSNCODE=*rsncode*]	*rsncode*: RS-type address or register (2) - (12).
[,PLISTVER=*plistver*|IMPLIED_VERSION]	**Default**: IMPLIED_VERSION
[,MF=S]	**Default**: MF=S
[,MF=(L,*mfctrl*,*mfattr*, 0D)]	
[,MF=(E,*mfctrl*,COMPLETE)]	

Parameters

In the following set of mutually exclusive keywords, only one keyword must be specified.

RETRIEVE
Retrieve SMF Recording information.

STATUS
Obtain information about the current log stream recording environment.

OUTAREA=*outarea*
A required character input/output specifying an area to contain the data being returned by IFAQUERY. The answer area is defined by the IFAQUAA macro, which consists of the QUAHDR and QUALSI structures. The IFAQUAA mapping macro provides the format of the area. The area can be in the primary address space or in an address space or data space that is addressable through a public entry on the caller's DU-AL. Use the OUTLEN parameter to specify the length of the area.

To code: Specify the RS-type address of a character field, or register (2) - (12) (ASM only).

OUTLEN=*outlen*
A required fullword input parameter that contains the length of the area provided to contain the data being returned by IFAQUERY.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) (ASM only).

End of the mutually exclusive keywords.

DETAILS=LOGSTREAM
An optional keyword input indicating the type of information that the SMF query service should return.

DEFAULT: LOGSTREAM.

RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) (ASM only).

RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or address in register (2) - (12) (ASM only).

PLISTVER=*plistver*|IMPLIED_VERSION
An optional byte input decimal value in the "0-0" range that specifies the macro version. PLISTVER is the only parameter allowed on the list form of MF. This parameter determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including

the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values can be:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently supported. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- 1

,MF=S
,MF=(L,*mfctrl*,*mfattr*, 0D)
MF=(E,*fctrl*,COMPLETE
An optional keyword input that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the servIice. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,fctrl
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

mfattr
An optional 1- 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning for each return and reason code.

Table 87. Return and Reason Codes for the IFAQUERY Macro

Return Code	Reason Code	Meaning and Action
00	None	**Explanation**: IFAQUERY request successful.
04		**Explanation**: Warning. Refer to the action provided with the specific reason code.
04	xxxxxx01	**Explanation**: OUTAREA is too small to contain all the requested data. The results in the OUTAREA were truncated. QUAHLEN specifies the amount of storage that is required to return a complete result. **Action:** Try the request again with a larger OUTAREA.
04	xxxxxx03	**Explanation**: DETAILS=LOGSTREAM was requested but no log stream information exists. The QUAHDR is filled in appropriately. No records are returned. **Action:** None.
08		**Explanation**: Incorrect input parameter. Refer to the action provided with the specific reason code.
08	xxxxxx01	**Explanation**: Caller was not running as a task. **Action:** Move the invocation of IFAQUERY under a task.
08	xxxxxx02	**Explanation**: The input parmlist cannot be accessed. **Action:** Check for one of the following possible errors: • Program exception during access of parameter list. • Parameter list has incorrect address.
08	xxxxxx03	**Explanation**: The QUAA area could not be accessed. **Action:** Check for one of the following possible errors: • Program exception during access of QUAA area. • QUAA area has incorrect address.
08	xxxxxx05	**Explanation**: The OUTAREA length is too small for a QUAA header. **Action:** Increase the size of the OUTAREA. The length must be greater than or equal to 16 bytes.
08	xxxxxx06	**Explanation**: QUAA has invalid ALET.
0C		**Explanation**: Environmental error. Refer to the action provided with the specific reason code.
0C	xxxxxx01	**Explanation**: SMF recording is not active. No records are returned. **Action:** None.

Table 87. Return and Reason Codes for the IFAQUERY Macro (continued)

Return Code	Reason Code	Meaning and Action
0C	xxxxxx02	**Explanation**: Storage for local area was not obtained. **Action:** None.
10	None	**Explanation**: Unexpected error. The state of the request is unpredictable.

Chapter 94. IFAWIC — IBM z/OS Workload Interaction Correlator

IBM z/OS Workload Interaction Correlator provides infrastructure to enable a z/OS component, middleware, or application product distributed across multiple address spaces to generate high-frequency (every 5 seconds), standardized, synchronized, contextualized activity using a common context. IBM z/OS Workload Interaction Correlator may be referred to herein as WIC.

z/OS Workload Interaction Correlator may be referred to herein as WIC.

Description

The IFAWIC service manages a z/OS component, middleware, or application as a WIC exploiter with the caller's primary address space and the system. The IFAWIC caller identifies itself by providing the IBM-assigned subtype.

The REGISTER request returns a buffer to the caller to use for instrumenting activities for the calling primary address space. The REGISTER request must be invoked in each address space in which a program runs. This supplies each address space with a distinct buffer for instrumenting address space activities.

The IFAWIC caller provides an exit routine that SMF will call during the WIC processing. This exit routine aggregates and summarizes activities from all address spaces that the exploiter registered. The exit routine also prepares and writes an SMF type 98 record for an exploiter's subtype.

The DEREGISTER request indicates that the program is no longer using the instrumentation buffer in the calling primary address space. When a program has been deregistered from all address spaces, the WIC exit routine will no longer be called and the system will clean it up.

Deregistration occurs implicitly at address space or job-step task termination.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be authorized with any of the following attributes: supervisor state, PKM 0-7, PSW key 0-7, or APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The control parameters must be accessible using the PSW key of the program making the request.

Programming requirements

The caller can include the IFAWICCB mapping macro to establish equate symbols for the return codes and reason codes provided by the IFAWIC macro.

The caller can listen for ENF event 85 for notifications about their subtype. See the IFAWICCB macro for details. The WicEnfQual section describes the QUAL input to the ENFREQ request. Use WicEnfQual_Subtype to get events for the exploiter's subtype only. WicEnf provides a mapping of the parameter list that is passed to the ENF listener exit in the first word of a 6-word parameter list in R1.

The first IFAWIC REQUEST=REGISTER caller on the system for a subtype must specify the ExitVersion and ExitRoutine parameters. This will establish the WIC exit routine to the program.

See IBM z/OS Workload Interaction Correlator in z/OS MVS Programming: Authorized Assembler Services Guide for details about how to code a WIC exit routine and how to use the WIC exit routine services.

Restrictions

Callers specifying REQUEST=REGISTER must not have functional recovery routines (FRRs).

Input register information

Before issuing the IFAWIC macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if register 15 contains a non-zero return code

1
Used as a work register by the system

2 - 13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0 - 1
Used as work registers by the system

2 - 13
Unchanged

14 - 15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

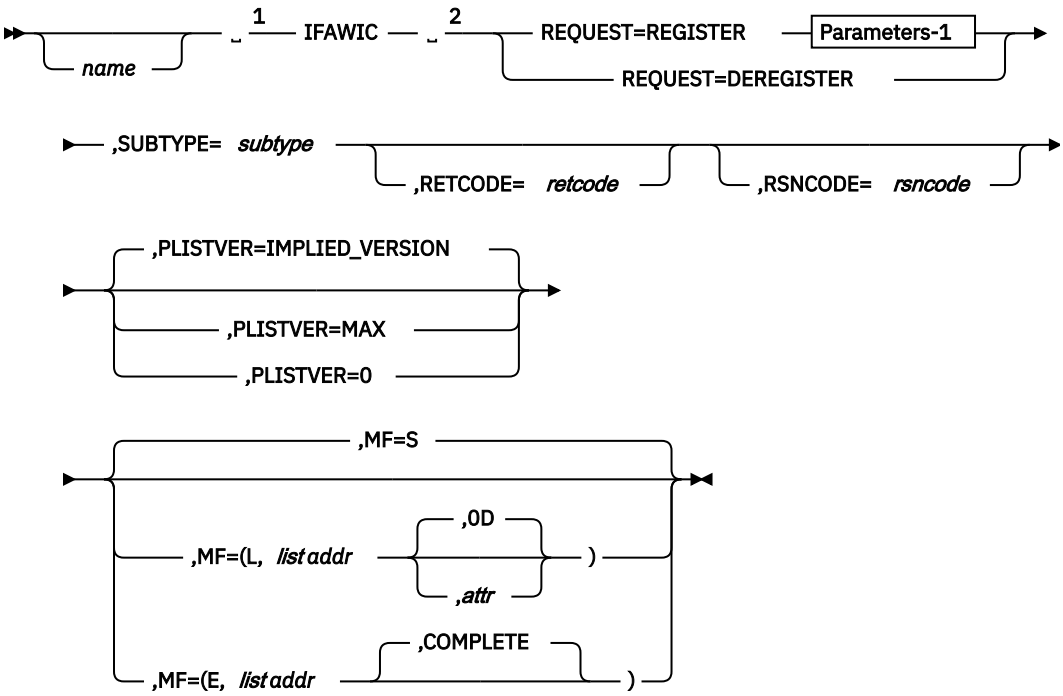
Performance implications

None.

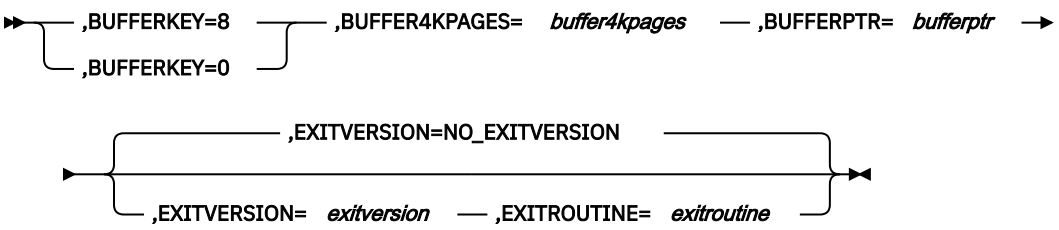
Syntax

The IFAWIC macro is written as follows:

Main diagram



Parameters-1



Notes:

[1] One or more blanks must precede the IFAWIC keyword.
[2] One or more blanks must follow the IFAWIC keyword.

Parameters

The IFAWIC parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IFAWIC macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=REGISTER
REQUEST=DEREGISTER
A required parameter that specifies the type of IFAWIC request.

REQUEST=REGISTER
Indicates a request to register the address space to the WIC instrumentation for the input subtype and to register the input subtype with the system.

When registering the address space for a subtype, the caller provides the desired characteristics of the WIC instrumentation buffer in the BUFFERKEY and BUFFER4KPAGES input fields. The buffer key must be the same across all register requests for the subtype. The caller is provided with a buffer addressable by the BUFFERPTR output field. The provided buffer can be used by the caller to save instrumentation data for the registering program for activity in calling address space.

The first IFAWIC call for a subtype must register the subtype with the system and provide a WIC exit routine by specifying the EXITVERSION and EXITROUTINE fields. Subsequent REGISTER requests may update the exit routine by specifying a higher exit version number.

REQUEST=DEREGISTER
Indicates that a program is no longer instrumenting into the buffer for the input subtype in the current primary address space.

The WIC instrumentation buffer for this subtype in this address space will become inaccessible, and the caller must have already stopped referencing this storage. The WIC instrumentation buffer for this address space will still be accessible to the WIC exit in the SMF address space, so ensuring the storage is accessible to the exit is not a concern to the caller.

When all address spaces registered for a program's subtype have deregistered, the system no longer calls the WIC exit routine. The WIC exit routine is cleaned up.

Deregistration occurs implicitly at address space or job-step task termination.

,BUFFERKEY=8
,BUFFERKEY=0
When REQUEST=REGISTER is specified, a required input parameter that specifies the storage key of the WIC instrumentation buffer to obtain. The caller must specify the same storage key for all register requests for the same subtype.

,BUFFERKEY=8
Indicates to obtain the instrumentation buffer in key 8.

,BUFFERKEY=0
Indicates to obtain the instrumentation buffer in key 0.

,BUFFER4KPAGES=*buffer4kpages*
When REQUEST=REGISTER is specified, a required input parameter that specifies the number of 4K pages in the WIC instrumentation buffer. This number must be 1 - 16 pages.

It is desirable for the BUFFER4KPAGES value to be the same for all register requests for the input subtype; however, the system allows different address spaces to request different buffer sizes for the same input subtype. When specifying different BUFFER4KPAGES values, the WIC exit routine must be able to determine how much data can be processed in each address space's WIC instrumentation buffer. Also, when aggregating data, new data fields may not be available for aggregation from older program versions. Generally, an old exit must be able to process data it is aware of from an old or new buffer, while a new exit must be able to process all data for all buffers.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,BUFFERPTR=*bufferptr*
When REQUEST=REGISTER is specified, a required output parameter that points to the page-aligned WIC instrumentation buffer for the input subtype for the current primary address space. This buffer must be accessed in AMODE 64.

To code: Specify the RS-type address, or address in register (2) - (12), of an eight-byte pointer field.

,EXITVERSION=*exitversion*
,EXITVERSION=NO_EXITVERSION
When REQUEST=REGISTER is specified, an optional input parameter that specifies the exit version number. When specified, the exit version must be an unsigned number greater than zero. The default is NO_EXITVERSION.

For the first REQUEST=REGISTER request or subsequent REQUEST=REGISTER requests where the exit version number is larger than previously requested for the input subtype, IFAWIC processing saves the input exit routine (*exitroutine*) and calls the exit routine on the next WIC recording interval.

When the EXITVERSION parameter is not specified or when EXITVERSION specifies a version number that is less than previously requested, IFAWIC processing ignores the EXITROUTINE parameter and continues to call the exit routine associated with the largest exit version number.

When the exit version number is equal to a previously specified request, IFAWIC processing validates that the specified exit routine is the same as the previously requested exit routine. If the exit routines are different, the IFAWIC request fails and the system continues to call the last successfully added exit routine.

In order to tell the system to use a new exit routine with the same or different name, specify a higher exit version number.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword field, or specify a literal decimal value.

,EXITROUTINE=*exitroutine*

When REQUEST=REGISTER and EXITVERSION=*exitversion* are specified, a required input parameter that specifies the name of the WIC exit routine to be given control in the SMF address space to generate SMF type 98 records for the input subtype.

The specified exit routine must be a valid module name residing in the calling program's TASKLIB, STEPLIB, or JOBLIB, the system's LNKLST concatenation, or LPA. The provided exit routine must be in an APF-authorized library and be re-entrant.

The exit routine must be in AMODE 31 or AMODE 64 and must access the WIC instrumentation buffers in AMODE 64.

To code: Specify the RS-type address, or address in register (2) - (12), of an 8-character field.

,SUBTYPE=*subtype*

A required input parameter that specifies the IBM-assigned subtype for writing SMF 98 records from WIC instrumentation data. The value must be a supported subtype and an unsigned number in the range 2 - 32767.

To code: Specify the RS-type address, or address in register (2) - (12), of a halfword field, or specify a literal decimal value.

,RETCODE=*retcode*

An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*

An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IFAWIC macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The IFAWICCB mapping macro provides equate symbols for the return and reason codes. Bits 0 - 15 of the reason code may contain component diagnostic data and must not be assumed to be 0. Logically AND the reason code with the IFAWIC_Rsncode_Mask mask in order to isolate the non-component diagnostic portion of the reason code.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the *xxxx* value.

Table 88. Return and reason codes for the IFAWIC service

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
0	–	**Equate symbol**: IFAWIC_Rc_Success **Meaning**: IFAWIC request successful. For REQUEST=REGISTER, a BufferPtr has been provided to the current primary address space. If an ExitVersion has been specified, and this is the first REQUEST=REGISTER on the system for the input subtype, or the input ExitVersion is larger than the current ExitVersion the input subtype, the ExitRoutine has been registered, and the system will switch to calling the new exit routine at the next WIC interval. The system issues ENF 85 signals to inform IFAWIC callers to start or stop writing instrumentation data to their instrumentation buffer. See macro IFAWICCB for additional information. For REQUEST=DEREGISTER, the buffer is made unavailable to the current primary address. By invoking REQUEST=DEREGISTER, the program certifies that the exploiter has stopped instrumenting into its buffer. The program does not have to certify its WIC exit is not accessing the buffer. Once REQUEST=DEREGISTER is invoked, the system is responsible for preventing the WIC exit from accessing the buffer If there are no more address spaces registered for the input subtype, the system will stop calling the exit routine. **Action**: For REQUEST=REGISTER, the program should begin instrumenting. For REQUEST=DEREGISTER, none.

Table 88. Return and reason codes for the IFAWIC service (continued)

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
4	–	**Equate symbol**: IFAWIC_Rc_Warning **Meaning**: IFAWIC request completed with a warning. The high half word of the reason code may contain indications of other warnings that have occurred besides the reason code portion in the lower half. See Wic_Warning_Rsn in IFAWICCB for a mapping of this area. For REQUEST=REGISTER, a BufferPtr has been provided to the current primary address space. If an ExitVersion has been specified, and this is the first REQUEST=REGISTER on the system for the input subtype, or the input ExitVersion is larger than the current ExitVersion the input subtype, the ExitRoutine has been registered, and the system will switch to calling the new exit routine at the next WIC interval. For REQUEST=DEREGISTER, the buffer is made unavailable to the current primary address. If there are no more address spaces registered for the input subtype, the system will stop calling the exit routine. **Action**: Refer to the action provided with the specific reason code. For REQUEST=REGISTER, the program should check the high half of the reason code for any conditions indicating the WIC services are not fully available by testing for any bits being on in the first byte using the IFAWICCB field Wic_NotFullyAvail_LowHighMask. When no bits in the first byte of the reason code are on, the program should start writing instrumentation data to the instrumentation buffer because the program's WIC exit will be called at the next WIC interval. When any bit in the first byte of the reason code is on, the program should not start writing instrumentation data to the instrumentation buffer because the WIC exit will not be called at the next WIC interval The system issues ENF 85 signals to inform IFAWIC callers to start or stop writing instrumentation data to their instrumentation buffer. See macro IFAWICCB for additional information. For REQUEST=DEREGISTER, none.

Table 88. Return and reason codes for the IFAWIC service (continued)

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
4	*xxxx*0401	**Equate symbol**: IFAWIC_Rsn_Buffer4kPagesSmaller **Meaning**: A smaller Buffer4kPages value was requested than the value specified on a previous REQUEST=REGISTER for the same subtype. The system reserves the requested buffer size for this address space. **Action**: It is desirable for the BUFFER4KPAGES value to be the same for all register requests for the input subtype; however, the system allows different address spaces to request different buffer sizes for the same input subtype. When specifying different BUFFER4KPAGES values, the WIC exit routine must be able to determine how much data can be processed in each address space's WIC instrumentation buffer. Also, when aggregating data, new data fields may not be available for aggregation from older program versions. Generally, an old exit must be able to process data it is aware of from an old or new buffer, while a new exit must be able to process all data for all buffers.
4	*xxxx*0402	**Equate symbol**: IFAWIC_Rsn_Buffer4kPagesLarger **Meaning**: A larger Buffer4kPages value was requested than the value specified on a previous REQUEST=REGISTER for the same subtype. The system reserves the requested buffer size for this address space. **Action**: It is desirable for the BUFFER4KPAGES value to be the same for all register requests for the input subtype; however, the system allows different address spaces to request different buffer sizes for the same input subtype. When specifying different BUFFER4KPAGES values, the WIC exit routine must be able to determine how much data can be processed in each address space's WIC instrumentation buffer. Also, when aggregating data, new data fields may not be available for aggregation from older program versions. Generally, an old exit must be able to process data it is aware of from an old or new buffer, while a new exit must be able to process all data for all buffers.
4	*xxxx*0403	**Equate symbol**: IFAWIC_Rsn_AlreadyRegistered **Meaning**: A REQUEST=REGISTER was requested from an address space for a subtype that was already registered. **Action**: none needed.
4	*xxxx*0404	**Equate symbol**: IFAWIC_Rsn_NotRegistered **Meaning**: A REQUEST=DEREGISTER was requested from an address space for a subtype that was not currently registered. **Action**: none needed.

Table 88. Return and reason codes for the IFAWIC service (continued)

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
4	*xxxx*0405	**Equate symbol**: IFAWIC_Rsn_ExitVersionIgnored **Meaning**: A REQUEST=REGISTER specified an ExitVersion that was less than the current ExitVersion. The ExitVersion and ExitRoutine parameters are ignored and the system continues using the current highest versioned exit routine. **Action**: none needed.
4	*xxxx*0481	**Equate symbol**: IFAWIC_Rsn_WicFeatureNotEnabled **Meaning**: The WorkloadIntCorr feature was not enabled to product registry. **Action**: Wait for ENF 85 signal and verify the WicEnf bit WicEnf_InstrumentationRequested is on before writing to the instrumentation buffer.
4	*xxxx*0482	**Equate symbol**: IFAWIC_Rsn_SubtypeNotCollected **Meaning**: SMF has not been configured to collect the input SMF 98 subtype record. **Action**: Wait for ENF85 signal and verify the WicEnf bit WicEnf_InstrumentationRequested is on before writing to the instrumentation buffer.
8	–	**Equate symbol**: IFAWIC_Rc_UserError **Meaning**: The IFAWIC request specified parameters that are not valid or the request is issued in an user-controllable environment that is not valid. The IFAWIC service did not complete successfully. For REQUEST=REGISTER, no WIC instrumentation buffer was provided to the caller. **Action**: Refer to the action provided with the specific reason code.
8	*xxxx*0801	**Equate symbol**: IFAWIC_Rsn_ParmListAlet **Meaning**: Unable to use ALET of the IFAWIC parameter list. **Action**: Provide a valid ALET for the parameter list. The access register might not have been set up correctly.
8	*xxxx*0802	**Equate symbol**: IFAWIC_Rsn_BadVersion **Meaning**: The provided PLISTVER is not valid. This suggests the parameter list used to call IFAWIC was overlaid, or was not generated by the IFAWIC macro. **Action**: Check for possible storage overlay of the parameter list. Use the IFAWIC service to produce the parameter list.

Table 88. Return and reason codes for the IFAWIC service (continued)

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
8	*xxxx*0803	**Equate symbol**: IFAWIC_Rsn_ReservedNot0 **Meaning**: The parameter list contains non-0 input in reserved fields. This suggests the parameter list used to call IFAWIC was overlaid, or was not generated by the IFAWIC macro. **Action**: Check for possible storage overlay of the parameter list. Use the IFAWIC service to produce the parameter list.
8	*xxxx*0804	**Equate symbol**: IFAWIC_Rsn_BadRequestType **Meaning**: The request type is not valid. This suggests the parameter list used to call IFAWIC was overlaid, or was not generated by the IFAWIC macro. **Action**: Check for possible storage overlay of the parameter list. Use the IFAWIC service to produce the parameter list.
8	*xxxx*0805	**Equate symbol**: IFAWIC_Rsn_BadBufferKey **Meaning**: The buffer key type is not valid. This suggests the parameter list used to call IFAWIC was overlaid, or was not generated by the IFAWIC macro. **Action**: Check for possible storage overlay of the parameter list. Use the IFAWIC service to produce the parameter list.
8	*xxxx*0807	**Equate symbol**: IFAWIC_Rsn_ParmListFetch **Meaning**: An error was encountered when fetching the supplied parameter list. **Action**: Call IFAWIC with a parameter list properly addressable.
8	*xxxx*0808	**Equate symbol**: IFAWIC_Rsn_ParmListWrite **Meaning**: An error was encountered when writing to the supplied parameter list. **Action**: Call IFAWIC with a parameter list properly addressable and able to be written to.
8	*xxxx*0810	**Equate symbol**: IFAWIC_Rsn_NotEnabled **Meaning**: The IFAWIC caller was not enabled. **Action**: Call IFAWIC only when enabled.
8	*xxxx*0811	**Equate symbol**: IFAWIC_Rsn_Locked **Meaning**: The IFAWIC caller was locked. **Action**: Call IFAWIC without holding locks.
8	*xxxx*0812	**Equate symbol**: IFAWIC_Rsn_HomeNotPrimary **Meaning**: The IFAWIC caller's primary home space was not equal to the caller's primary address space. **Action**: Call IFAWIC when home equals primary address space.

Table 88. Return and reason codes for the IFAWIC service (continued)

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
8	*xxxx*0813	**Equate symbol**: IFAWIC_Rsn_CallerFRR **Meaning**: The IFAWIC REQUEST=REGISTER caller had FRR recovery established. **Action**: Call IFAWIC without FRRs established.
8	*xxxx*0814	**Equate symbol**: IFAWIC_Rsn_NotAuthorized **Meaning**: The IFAWIC caller is not authorized. **Action**: Call IFAWIC only when authorized.
8	*xxxx*0820	**Equate symbol**: IFAWIC_Rsn_SubtypeInput **Meaning**: The provided SUBTYPE is not supported. **Action**: Supply the IBM provided program subtype.
8	*xxxx*0821	**Equate symbol**: IFAWIC_Rsn_ExitVersionZero **Meaning**: The ExitVersion input must not be 0. **Action**: Specify an ExitVersion that is not 0.
8	*xxxx*0822	**Equate symbol**: IFAWIC_Rsn_Buffer4kPagesZero **Meaning**: The Buffer4kPages must not be 0. **Action**: Specify Buffer4kPages that is not 0.
8	*xxxx*0823	**Equate symbol**: IFAWIC_Rsn_Buffer4kPagesTooLarge **Meaning**: The Buffer4kPages input must be less than or equal to 16. **Action**: Specify Buffer4kPages as less than or equal to 16.
8	*xxxx*0824	**Equate symbol**: IFAWIC_Rsn_BufferKeyMismatch **Meaning**: A previous REQUEST=REGISTER requested and received a buffer for a BUFFERKEY value that was different that the BUFFERKEY specified on this request. **Action**: Update the program to request the same BUFFERKEY for each call to IFAWIC, even across different program versions.
8	*xxxx*0825	**Equate symbol**: IFAWIC_Rsn_SRBMode **Meaning**: IFAWIC was issued in SRB mode **Action**: Do not issue IFAWIC in SRB mode
8	*xxxx*0830	**Equate symbol**: IFAWIC_Rsn_SameVerExitRoutineMismatch **Meaning**: A ExitRoutine and ExitVersion was specified such that the ExitVersion is the same as a previous specification, however, the ExitRoutine is different. **Action**: Specify the same ExitRoutine to correspond with the same ExitVersion. If the program introduces a new exit routine, the ExitVersion must be made larger.

Table 88. Return and reason codes for the IFAWIC service (continued)

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
8	*xxxx*0831	**Equate symbol**: IFAWIC_Rsn_ExitRtnNotFound **Meaning**: The exit routine could not be found. **Action**: Ensure the exit routine is located in either the invoking program's JOBLIB, STEPLIB, TASKLIB, in the system's LNKLST concatenation, or in LPA.
8	*xxxx*0832	**Equate symbol**: IFAWIC_Rsn_ExitRtnNotInAPFLib **Meaning**: IFAWIC was unable to load the provided exit routine because did not reside in an APF authorized library. The *xxxx* portion of the reason code contains an informational code as documented with ABEND 306 reasons. No abend was generated. **Action**: See *z/OS MVS System Codes*, sections "System completion codes," "306," with the *xxxx* portion of the reason code to see what action is needed to resolve the condition. Ensure the exit routine resides in an APF-authorized library and is re-entrant.
8	*xxxx*0833	**Equate symbol**: IFAWIC_Rsn_ExitRtnAmode24 **Meaning**: The ExitRoutine must not be in AMODE 24. **Action**: Provide an exit routine with an AMODE of 31 or 64.
8	*xxxx*0834	**Equate symbol**: IFAWIC_Rsn_ExitRtnNotReEntrant **Meaning**: The ExitRoutine must be re-entrant. **Action**: Provide an exit routine that is linked as re-entrant.
8	*xxxx*0835	**Equate symbol**: IFAWIC_Rsn_ExitRtnNameNotUnique **Meaning**: Another subtype is using the same exit routine name. The exit routine name must be unique per subtype. **Action**: Provide a unique exit routine name.
C	–	**Equate symbol**: IFAWIC_Rc_EnvError **Meaning**: Environmental error The IFAWIC service did not complete successfully. For REQUEST=REGISTER, no WIC instrumentation buffer was provided to the caller. **Action**: Refer to the action provided with the specific reason code.
C	*xxxx*0C01	**Equate symbol**: IFAWIC_Rsn_SystemNotReady **Meaning**: IFAWIC was issued before the system is ready to start processing requests. **Action**: Wait for the system to be ready to accept IFAWIC requests and re-issue the IFAWIC request.

Table 88. Return and reason codes for the IFAWIC service (continued)

Return code (hex)	Reason code (hex)	Equate symbol, meaning, and action
C	*xxxx*0C02	**Equate symbol**: IFAWIC_Rsn_UnsupportedMachine **Meaning**: IFAWIC was issued from an unsupported machine. WIC requires IBM z14 or later hardware. **Action**: Issue IFAWIC on a supported machine.
C	*xxxx*0C03	**Equate symbol**: IFAWIC_Rsn_EnvNoExit **Meaning**: IFAWIC REQUEST=REGISTER requests omitting ExitVersion and ExitRoutine, require a WIC exit routine to have been already established by a previous IFAWIC REQUEST=REGISTER call specifying the ExitVersion and ExitRoutine parameters. **Action**: Specify ExitVersion and ExitRoutine or ensure an IFAWIC REQUEST=REGISTER call has already completed specifying ExitVersion and ExitRoutine before issuing IFAWIC REQUEST=REGISTER.
C	*xxxx*0C04	**Equate symbol**: IFAWIC_Rsn_SMFNOWICSpecified **Meaning**: SMF parameters specified NOWIC which prevents programs from issuing IFAWIC REQUEST=REGISTER. **Action**: Accept that WIC services will be unavailable for the program, or choose to handle WIC registration dynamically. If the program can support dynamic WIC registration, wait for ENF 85 signal and check the WicEnf bit WicEnf_RegisterIsAvailable. When WicEnf_RegisterIsAvailable is on, The ENF listener exit can cause a task to wake up to re-issue IFAWIC REQUEST=REGISTER for the program. The system issues ENF 85 signals to inform IFAWIC callers that IFAWIC Register is available when SMF parameter WIC is specified. See macro IFAWICCB for additional information.
C	*xxxx*0C10	**Equate symbol**: IFAWIC_Rsn_ExitRtnNoStorage **Meaning**: There was not sufficient storage for IFAWIC to process the exit routine. **Action**: Contact your system programmer. there is a shortage of common storage.
C	*xxxx*0C11	**Equate symbol**: IFAWIC_Rsn_UnexpectedLoadError **Meaning**: IFAWIC encountered errors trying to load the provided exit routine. The "xxxx" portion of the reason code contains a code from the load service of what would have been the ABEND code. No abend was generated. **Action**: Alert the system programmer. See *z/OS MVS System Codes*, sections "System completion codes," with the "xxxx" portion as the abend code to see what action is needed to resolve the condition. This may be accompanied by system log messages with the CSV prefix.

Table 88. Return and reason codes for the IFAWIC service (continued)		
Return code (hex)	**Reason code (hex)**	**Equate symbol, meaning, and action**
10	–	**Equate symbol**: IFAWIC_Rc_CompError **Meaning**: Unexpected failure. **Action**: Refer to the action provided with the specific reason code.
10	*xxxx*1001	**Equate symbol**: IFAWIC_Rsn_CompError **Meaning**: Unexpected failure. The state of the request is unpredictable. **Action**: Contact your system programer to report the problem to IBM service

Example

This example performs the following operations:

1. Issue the IFAWIC service to start WIC instrumentation for a test subtype.
2. Issue the IFAWIC service to end WIC instrumentation for a test subtype.

The code is as follows:

```
 ... other code here ...

         LARL  13,Wic_Fields
         USING Wic_Fields,13
         ENFREQ ACTION=LISTEN,CODE=ENFPC085,EOM=YES,         *
               SRBEXIT=MyWicEnfExit,                         *
               QUAL=MySubtype,QMASK=(BYTE3,BYTE4),           *
               DTOKEN=MyWicEnfToken Listen for ENF 85
         IFAWIC REQUEST=REGISTER,                            *
               SUBTYPE=MySubtype,                            *
               ExitRoutine=MyExitRtn,                        *
               ExitVersion=MyExitVer,                        *
               Buffer4kPages=MyBuf4kPg,                      *
               BUFFERKEY=8,                                  *
               BUFFERPTR=MyBuffer@  Register WIC subtype
         CFI  15,IFAWIC_Rc_Warning  Check for a warning RC
         JH   WicRegFailed          RC>4 is an error
         JL   WicRegSuccess         RC=0 is a success
         TMLH 0,Wic_NotFullyAvail_LowHighMask WIC services   *
                                    fully available?
         JZ   WicRegSuccess         Yes, instrument
         J    WicRegDone            No, do not instrument
WicRegFailed  DS 0H                 Register RC > 4
         NI   MyWicFlags,MyWicRegisterFailed remember failed
         ENFREQ ACTION=DELETE,CODE=ENFPC085,                 *
               DTOKEN=MyWicEnfToken Stop listening to the ENF
         J    WicRegDone
WicRegSuccess DS 0H                 Register succeeded
         NI   MyWicEnfFlags,WicEnf_InstrumentationRequested  *
                                    Indicate to write        *
                                    instrumentation data
WicRegDone DS 0H
*
* Place additional return/ reason code checking here
*

 ... other code here ...

         TM   MyWicEnfFlags,WicEnf_InstrumentationRequested  *
                                    Instrumentation ok now?
         JZ   InstrumentDone        No, do not instrument

 ... code using the provided buffer here ...

InstrumentDone DS 0H                After WIC instrumentation

 ... other code here ...
```

```
         TM   MyWicFlags,MyWicRegisterFailed WIC Reg fail?
         JO   WicDeregDone          Skip dereg if reg failed
         IFAWIC REQUEST=DEREGISTER,                          *
               SUBTYPE=MySubtype    Deregister WIC subtype
         ENFREQ ACTION=DELETE,CODE=ENFPC085,                 *
               DTOKEN=MyWicEnfToken Stop listening to the ENF
*
* Place code to check return/reason codes here
*
WicDeregDone DS 0H
 ... other code here ...
WICENFXT DS 0H                      Entry to SRB
         LARL 15,Wic_Fields         Load the working storage ptr
         USING Wic_Fields,15
         L    1,0(1)                Load WicEnf Parmlist
         USING WICENF,1
         IC   0,WicEnf_Flags        Copy the WicEnf_Flags
         STC  0,MyWicEnfFlags       Save in MyWicEnfFlags
         BR   14                    Return from SRB

 ... other code here ...
Wic_Fields DS 0D
MyBuffer@  DS D
MyExitRtn  DC CL8'WICEXIT '
MySubtype  DC F'24576'
MyExitVer  DC F'1'
MyBuf4kPg  DC F'4'
MyWicEnfToken DS F
MyWicEnfExit DC A(WICENFXT)
MyWicFlags DC X'00'
MyWicRegisterFailed EQU X'80'
MyWicEnfFlags DC X'00'


 ... other code here ...
```

Chapter 95. IOCINFO — Obtain MVS I/O configuration information

Description

Use the IOCINFO macro to obtain the following I/O configuration information:

- I/O configuration token
- Default channel subsystem identifier for the logical partition
- The maximum device measurement block index that is currently assigned
- The I/O facilities that are supported and enabled by the hardware and software.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key. For LINKAGE=BRANCH, all of the following: • Supervisor state with PSW key 0 • 31-bit addressing mode • Primary ASC mode • Parameter list and any data areas it points to must be in fixed storage or, if the caller is disabled, in disabled reference (DREF) storage
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- bit
ASC mode:	Primary or access register (AR)
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any
Control parameters:	Must be in the primary address space or be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If in AR mode, specify SYSSTATE ASCENV=AR before invoking the macro.

Restrictions

None.

Input register information

Before issuing the IOCINFO macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code if GPR 15 contains a return code of 08; otherwise, used as a work register by the system

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The standard form of the IOCINFO macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOCINFO.
IOCINFO	
ƀ	One or more blanks must follow IOCINFO.
IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).

Syntax	Description
,DCMINFO=*xdcminfo*	*xdcminfo*: RS-type address or register (2) - (12).
,CSSID=*cssid addr*	*cssid addr*: RX-type address or register (2) - (12).
,MAXMBI=*maxmbi addr*	*maxmbi addr*: RS-type address or register (2) - (12).
,IOFACILITIES=*iofc addr*	*iofc addr*: RX-type address or register (2) - (12).
,IODFINFO=*xiodfinfo*	*xiodfinfo*: RS-type address or register (2) - (12).
,LINKAGE=SYSTEM	**Default**: SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).
,PLISTVER=*xplistver*	
,PLISTVER=IMPLIED_VERSION	Default: IMPLIED_VERSION

Parameters

The parameters are explained as follows:

IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character area where the system returns the current MVS I/O configuration token.

,DCMINFO=*xdcminfo*
Specifies the address of an optional 32 character output area into which IOCINFO is to return Dynamic Channel Path Management (DCM) information which can be mapped by IOSDDCMI.

,CSSID=*cssid addr*
Specifies the address of a one byte output area where the system returns the default channel subsystem ID for the logical partition.

- A return code of X'00', reason code of X'00' indicates that the program is running on a processor that supports multiple channel subsystems.
- A return code of X'00', reason code X'01' indicates that the program is running on a processor that does not support multiple channel subsystems, and the CSS ID assigned is a zero.

,MAXMBI=*maxmbi addr*
Specifies the address of a halfword field where the system returns the maximum device measurement block index that is currently assigned.

,IOFACILITIES=*iofc addr*
Specifies the address of a required 256-byte output area into which the IOCINFO service returns the I/O facility information. This area is mapped by mapping macro IOSDIOFC.

,IODFINFO=*xiodfinfo*
Specifies the address of an optional 128 character output area into which IOCINFO is to return IODF information which is mapped by IOSDIODI.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Specifies the type of call that should be generated:

- **SYSTEM**: Specifies a Program Call (PC)
- **BRANCH**: Specifies a branch entry

LINKAGE=BRANCH is intended for performance-sensitive programs.

,RETCODE=*retcode addr*
Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsncode addr*
Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0.

,PLISTVER=*xplistver*
,PLISTVER=IMPLIED_VERSION
An optional byte input decimal value (with a value of 1) that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number, and the parameter list will be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

ABEND codes

None.

Return and reason codes

When the system returns control to the caller, GPR 15 (and *retcode addr*, if you coded RETCODE) contains the return code. For return code X'08', the reason code is in GPR 0 (and *rsncode addr*, if you coded RSNCODE).

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00		**Meaning**: Successful completion. **Action**: None.
00	00	**Meaning**: Successful completion from a CSSID parameter request. The program is running on a processor that supports multiple channel subsystems. **Action**: None.
00	01	**Meaning**: Successful completion from a CSSID parameter request. The program is running on a processor that does not support multiple channel subsystems and the CSS ID assigned is a zero. **Action**: None.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	01	**Meaning**: Program error. An ALET in the parameter list is not valid. The caller might have inadvertently written over an area in the parameter list. **Action**: Check to see if your program inadvertently overlaid the parameter list generated by the macro.
08	02	**Meaning**: Program error. The system could not access the caller's parameter list. **Action**: Check to see if your program inadvertently overlaid the parameter list generated by the macro.
08	05	**Meaning**: Program error. An error occurred when the system referenced the user-supplied area specified in the IOCTOKEN parameter. **Action**: Check to see if your program correctly specified the IOCTOKEN area.
08	09	**Meaning**: System error. This reason code is for IBM diagnostic purposes only. **Action**: Record the reason code and supply it to the appropriate IBM support personnel.
08	0F	**Meaning**: An error occurred referencing the user-supplied area that is specified in the IOFACILITIES parameter. **Action**: Check to see if your program correctly specified the IOFACILITIES area.
20		**Meaning**: System error. This return code is for IBM diagnostic purposes only. **Action**: Record the return code and supply it to the appropriate IBM support personnel.
24	07	**Meaning**: Program error. The system does not support the specified parameter. **Action**: Check the parameters on the IOCINFO macro to make sure they are valid on your release of the system.

IOCINFO—List form

Use the list form of the IOCINFO macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

Syntax

The list form of the IOCINFO macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.

Syntax	Description
ƀ	One or more blanks must precede IOCINFO.
IOCINFO	
ƀ	One or more blanks must follow IOCINFO.
MF=(L,*list addr*)	*list addr*: Symbol.
MF=(L,*list addr*,*attr*)	*attr*: 1- to 60- character input string
MF=(L,*list addr*,0D)	**Default**: 0D

Parameters

The parameters are explained under the standard form of the IOCINFO macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr*,*attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOCINFO macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IOCINFO - Execute form

Use the execute form of the IOCINFO macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOCINFO macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOCINFO.
IOCINFO	
ƀ	One or more blanks must follow IOCINFO.

Syntax	Description
IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,CSSID=*cssid addr*	*cssid addr*: RS-type address or register (2) - (12).
,MAXMBI=*maxmbi addr*	*maxmbi addr*: RS-type address or register (2) - (12).
,IOFACILITIES=*iofc addr*	*iofc addr*: RS-type address or register (2) - (12).
,LINKAGE=SYSTEM	**Default**: SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE

Parameters

The parameters are explained under the standard form of the IOCINFO macro with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the IOCINFO macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

Chapter 96. IOSADMF — Transfer hiperspace data

Description

The IOSADMF macro provides an interface for the movement of large amounts of data between main and expanded storage.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or program key mask (PKM) allowing keys 0-7.
Dispatchable unit mode:	Task or SRB mode for AREAD, AWRITE, and AQUERY requests. Task mode only for APURGE requests.
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

The caller's parameter list and range list must be in the primary address space.

Restrictions

For IOSADMF APURGE requests, the caller may not have an EUT FRR established.

Input register information

Before issuing the IOSADMF macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system.

2-14
Unchanged

15
Return code

When control returns to the caller, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

Using IOSADMF to move large amounts of data between central and expanded storage is more efficient than synchronous methods of moving data.

Syntax

The standard form of the IOSADMF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSADMF.
IOSADMF	
ƀ	One or more blanks must follow IOSADMF.
APURGE	
AREAD	
AWRITE	
AQUERY	
,ALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=*n*	*n*: Number from 1 to 125.
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12). **Default**: NUMRANGE=1.

Syntax	Description
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).
,FAILBLKP=*fail-addr*	*fail-addr*: RX-type address or register (2) - (12).
,CROSSOVER=*cross-addr*	*cross-addr*: RX-type address or register (2) - (12).
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).
,MF=S	

Table 89. Parameters Valid with IOSADMF Requests

Parameters	APURGE	AREAD	AWRITE	AQUERY
ALET	required	required	required	not valid
NUMRANGE	not valid	optional	optional	not valid
RANGLIST	not valid	required	required	not valid
FAILBLKP	not valid	optional	optional	not valid
CROSSOVER	not valid	not valid	not valid	optional
RETCODE	optional	optional	optional	optional
RSNCODE	optional	optional	optional	optional
MF	optional	optional	optional	optional

Parameters

The parameters are explained as follows:

APURGE
AREAD
AWRITE
AQUERY

Specifies the type of request, as follows: APURGE requests that the system purge any active AREAD or AWRITE operation for the hiperspace represented by the ALET.

AREAD requests that the system transfer data from a hiperspace to an address space.

AWRITE requests that the system transfer data from an address space to a hiperspace.

AQUERY requests that the system check to determine whether ADMF (asynchronous data mover facility) is installed. If ADMF is installed, the system returns a return code of 0. If ADMF is not installed, the system returns a return code of 8 with a corresponding reason code.

,ALET=*alet-addr*
Specifies either the address of a fullword or a register that contains the ALET associated with the hiperspace that is the target of an APURGE, AREAD, or AWRITE request.

,NUMRANGE=*n*
,NUMRANGE=*num-addr*
Specifies the number of entries in the range list in one of the following formats:

- A decimal digit from 1 through 125
- A fullword that contains the number of entries
- A register that contains the address of a fullword that contains the number of entries.

The default is NUMRANGE=1.

,RANGLIST=*list-addr*
Specifies a fullword that contains the address of a list of ranges (up to 125), or specifies a register that contains the address of the fullword pointer to the range list. The list of ranges specifies one or more virtual storage ranges that are to be moved. The range list consists of a number of entries (specified by NUMRANGE), where each entry consists of three words:

First word
The starting virtual address in the address space into which the data is to be read or from which data is to be written.

Second word
The starting virtual address in the hiperspace from which the system is to read or into which the system is to read.

Third word
The number of blocks the system is to read from the hiperspace or write from the address space.

For example, if your register and storage are set up as in Figure 9 on page 980, you can code the RANGLIST parameter and NUMRANGE parameter as follows:

```
NUMRANGE=3 ,RANGLIST=(5)

        or

NUMRANGE=3, RANGLIST=RANGADDR
```

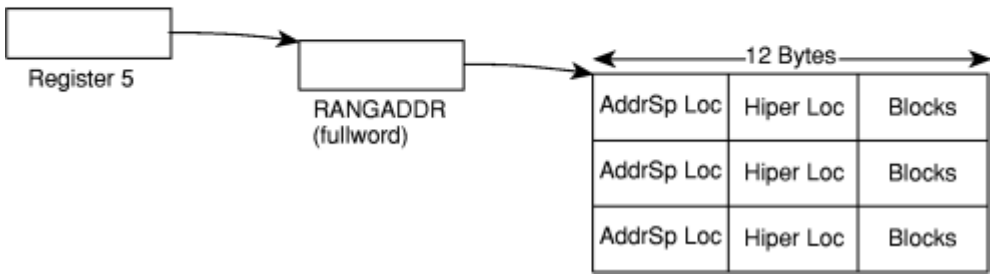


Figure 9. RANGLIST and NUMRANGE Parameters

The range list must be in the caller's primary address space.

,FAILBLKP=*fail-addr*
Specifies a fullword that contains the address of a range list entry, or specifies a register that contains the address of the fullword pointer to a range list entry, for which a failure occurred. The system returns this value only when you code FAILBLKP and when the system can identify the failing range list entry.

When the system returns a return code 8 and *fail-addr* contains a non-zero value, the entry identified by *fail-addr* is either partially processed, or not processed and any subsequent range list entries are not processed. However, any prior range list entries processed successfully.

fail-addr contains a non-zero value only when the failing range list is known. The reason codes indicate when *fail-addr* is set.

,CROSSOVER=*cross-addr*
Specifies a fullword or register in which the system is to place the system-implemented crossover value. If the number of pages requested to be moved is greater than the CROSSOVER value, the system moves the data asynchronously with the ADMF. If you invoke IOSADMF when the number of pages is less than the crossover value, the system uses the move page facility to move the data.

You can request this value to determine whether using the ADMF is warranted for particular data movement.

,RETCODE=*ret-addr*
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*rsn-addr*
Specifies the location where the system is to store the return code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the macro service.

ABEND codes

None.

Return and reason codes

When the IOSADMF macro returns control to the caller, GPR 15 (and *ret-addr*, if you coded RETCODE) contains a return code and GPR 0 (and *rsn-addr*, if you coded RSNCODE) contains a reason code.

The reason code consists of four bytes; the third byte contains a value that indicates where the error occurred. The third byte contains X'01' when the error occurred in an address space; it contains X'02' when the error occurred in a hiperspace.

Table 90. Return and Reason Codes for the IOSADMF Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	—	**Meaning**: The IOSADMF operation completed successfully. For an AQUERY request, return code 0 indicates that the ADMF is installed. **Action**: None.
04	xx0101xx	**Meaning**: System error. The IOSADMF operation failed because of a communication error. The request was started, but the system stopped the request because of an error condition. The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the address of the range list entry for which the failure occurred. **Action**: Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support.
04	xx0202xx	**Meaning**: System error. The IOSADMF operation failed because of a communication error. The request was started, but the system stopped the request because of an error condition. The failure occurred in the storage whose address is in the second word of the input range list entry. FAILBLKP contains the address of the range list entry for which the failure occurred. **Action**: Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support.

Table 90. Return and Reason Codes for the IOSADMF Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
04	xx0301xx	**Meaning**: Program error. A specified address identified an area in an address space for which the caller is not authorized. A protection exception was encountered. The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the address range list entry for which the failure occurred. **Action**: Either specify the address of an address space that the user has the authority to access, or obtain adequate authority to use the specified address. Retry the operation using IOSADMF or use the HSPSERV macro.
04	xx0501xx	**Meaning**: Program error. An error occurred during address translation. The request cannot be completed at the current time because an address space page was not valid. Either the address in the first word of the input range list entry was not correct or identified an area that was not backed. The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the range list entry for which the failure occurred. **Action**: Either retry the operation using IOSADMF or use the HSPSERV macro. Ensure that all the pages that are to be used are page fixed.
04	xx0502xx	**Meaning**: Program error. An error occurred during address translation. The request cannot be completed at the current time because a hiperspace page was not valid. Either the hiperspace in the second word of the input range list entry was not correct or identified an area that was reclaimed by the system. The failure occurred in the storage area whose address is in the second word of the input range list entry. FAILBLKP contains the address of a range list entry for which the failure occurred. **Action**: Use the HSPSERV macro to restore the hiperspace page.
04	xx0601xx	**Meaning**: System error. An uncorrectable storage error occurred at either the source or destination of the data move. The failure occurred in the storage area whose address is in the first word of the input range list entry. FAILBLKP contains the range list entry for which the failure occurred. **Action**: Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support.
04	xx0702xx	**Meaning**: System error. An uncorrectable storage error occurred at either the source or destination of the data move. The failure occurred in the storage area whose address is in the second word of the input range list entry. FAILBLKP contains the range list entry for which the failure occurred. **Action**: Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support.
04	xx0Cxxxx	**Meaning**: System error. An uncorrectable storage error occurred at either the source or destination of the data move. The system could not determine whether the error occurred in the address space storage or the hiperspace storage. FAILBLKP contains the range list entry for which the failure occurred. **Action**: Either retry the operation using IOSADMF or use the HSPSERV macro. If you still get the same error, record the return and reason codes; contact hardware support.
08	xx31xxxx	**Meaning**: Environmental error. The ADMF is not installed on the current system. The ADMF cannot be used until both hardware and software are installed and the operating system is IPLed. **Action**: Retry the operation using the HSPSERV macro instead of the IOSADMF macro.

Table 90. Return and Reason Codes for the IOSADMF Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	xx32xxxx	**Meaning**: System error. The asynchronous data mover facility is not available. The system detected an unrecoverable error. **Action**: Use the HSPSERV macro instead of IOSADMF and rerun the program. Record the return and reason codes; contact hardware support.
08	xx34xxxx	**Meaning**: Program error. The calling program does not meet one or more of the environmental requirements for using IOSADMF. **Action**: Ensure that the IOSADMF macro is issued in the required environment. See in "Environment" on page 977.
08	xx35xxx	**Meaning**: Program error. Either no option (AWRITE, AREAD, APURGE, or AQUERY) was specified on the IOSADMF macro, or more than one option was specified. This error can occur if the parameter list is overlaid. **Action**: Make sure the IOSADMF macro invocation specifies one option and rerun the program.
08	xx36xxxx	**Meaning**: Program error. The specified ALET is incorrect. The ALET did not designate a hiperspace, or the ALET is not on the caller's access list. **Action**: Make sure the ALET is valid and rerun the program.
08	xx37xxxx	**Meaning**: Program error. The range count is not valid. The NUMRANGE value specified is either less than 1 or greater than 125. **Action**: Specify a NUMRANGE value from 1 through 125 and rerun the program.
08	xx38xxxx	**Meaning**: Program error. An input parameter list could not be addressed, or an error occurred during a reference to a range list entry. The RANGLIST parameter may be specified incorrectly. **Action**: Ensure RANGLIST is specified correctly, and NUMRANGE is a valid value, and rerun the program.
08	xx39xxxx	**Meaning**: Program error. An error occurred during the processing of a RANGLIST entry address. FAILBLKP contains the address of the failing entry. **Action**: Ensure the following: • The RANGLIST parameter is correctly specified • The address on the RANGLIST parameter is correct • The NUMRANGE value reflects the actual number of NUMRANGE entries. • The NUMRANGE value is from 1 though 125. Rerun the program.
08	xx3Axxxx	**Meaning**: System error. This return and reason code combination is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply them to the IBM Support Center.
08	xx3Bxxxx	**Meaning**: Program error. The calling program does not meet one or more of the environmental requirements for using IOSADMF. **Action**: Ensure IOSADMF is issued in the required environment. See in "Environment" on page 977.
08	xx3Cxxxx	**Meaning**: Program error. An incorrect version of the ADMF was specified. The current version is 1. This error can occur if the parameter list is overlaid. **Action**: Contact your software support.
08	xx3Exxxx	**Meaning**: Program error. The reserved fields in XFLAGS, XRESERVED1, or XRESERVED2 are not zero. These fields must be set to zero before the IOSADMF macro can be invoked. **Action**: See the IOSADMF macro expansion. Correct the parameter list and rerun the program.

Table 90. Return and Reason Codes for the IOSADMF Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	xx40xxxx	**Meaning**: Program error. The caller attempted to access a hiperspace using the IOSADMF macro, but the hiperspace is in the process of being deleted. The access request is rejected. **Action**: Specify the ALET of another hiperspace and reissue the IOSADMF request.
08	xx41xxxx	**Meaning**: System error. This return and reason code combination is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply them to the IBM Support Center.
0C	xx51xxxx	**Meaning**: System error. This return and reason code combination is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply them to the IBM Support Center.
0C	xx52xxxx	**Meaning**: System error. This return and reason code combination is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply them to the IBM Support Center.
0C	xx53xxxx	**Meaning**: System error. This return and reason code combination is for IBM diagnostic purposes only. **Action**: Record the return and reason codes and supply them to the IBM Support Center.

IOSADMF - List form

Use the list form of the IOSADMF macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

Syntax

The list form of the IOSADMF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSADMF.
IOSADMF	
ƀ	One or more blanks must follow IOSADMF.
,MF=(L,*list addr*)	*list addr*: symbol.
,MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
,MF=(L,*list addr*,0D)	**Default**: 0D

Syntax	Description

Parameters

The parameters are explained under the standard form of the SAMPLE macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOSADMF macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IOSADMF - Execute form

Use the execute form of the IOSADMF macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSADMF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSADMF.
IOSADMF	
ƀ	One or more blanks must follow IOSADMF.
APURGE	
AREAD	
AWRITE	
AQUERY	
,ALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=*n*	*n*: Decimal digit from 1 to 125.

Syntax	Description
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12).
	Default: NUMRANGE=1.
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).
,FAILBLKP=*fail-addr*	*fail-addr*: RX-type address or register (2) - (12).
,CROSSOVER=*cross-addr*	*cross-addr*: RX-type address or register (2) - (12).
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE

Parameters

The parameters are explained under the standard form of the SAMPLE macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the IOSADMF macro.

list addr specifies the area that the system uses to store the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.

Chapter 97. IOSCAPF — Obtain the actual address of a captured UCB

Description

Use the IOSCAPF macro to obtain the actual address of a specified captured unit control block (UCB) address. A captured UCB is a below 16 megabyte view of an above 16 megabyte UCB. The IOSCAPU macro performs the same function and provides input parameter validation, recovery, and environmental checking. IOSCAPF provides an alternative for passing parameters (that is, in register 1 rather than in a parameter list). IOSCAPU enables you to specify the address of a UCB in another address space. With IOSCAPF, the specified UCB must reside in the current address space.

For information about accessing an above 16 megabyte UCB, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key for READ type calls; Supervisor state and any PSW key for CREATE, UPDATE, and DELETE calls.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	None.

Programming requirements

The caller must pin the UCB or otherwise guarantee that the UCB will not be deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

Only use IOSCAPF to translate a captured UCB address that was captured in your primary address space.

Input register information

Before issuing the IOSCAPF macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of UCB common segment of the captured UCB

Before issuing the IOSCAPF macro, the caller does not have to place any information into any access register (AR).

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the UCB common segment of the actual UCB

2-13
Unchanged

14-15
Used as work registers by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IOSCAPF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPF.
IOSCAPF	
ƀ	One or more blanks must follow IOSCAPF.
MF=(S)	**Default**: S

Parameters

The parameters are explained as follows:

MF=(S)
Specifies the standard form of the macro. This parameter is optional.

ABEND codes

None.

Return and reason codes

None.

Chapter 98. IOSCAPU — Capture, release, or obtain the actual address of a UCB

Description

Use the IOSCAPU macro to access an above 16 megabyte unit control block (UCB) with a 24-bit address. IOSCAPU creates a view into the actual above 16 megabyte UCB in below 16 megabyte private storage, which is known as capturing the UCB. An above 16 megabyte UCB is automatically captured at allocation and released at deallocation. You can also use IOSCAPU to explicitly capture and release an above 16 megabyte UCB if necessary.

IOSCAPU enables you to perform the following functions:

- Capture an actual UCB into the private storage area of an address space and receive the captured UCB address with the CAPTUCB option. You can also capture the UCB into common storage.
- Release a captured UCB at a specific address with the UCAPTUCB option
- Receive the 31-bit above 16 megabyte actual address for a specified captured address with the CAPTOACT option.

The environment, programming requirements, restrictions, input register information, output register information, and performance implications generally apply to all the functions. Any exceptions are noted. The syntax, return and reason codes, abend codes, examples, and forms are described in a separate section for each function. See “Capture an UCB function” on page 993, “Release a captured UCB function” on page 998, and “Translate captured to actual address function” on page 1002.

Similar to IOSCAPU with the CAPTOACT option, the IOSCAPF macro obtains the above 16 megabyte address of a captured UCB. IOSCAPF enables you to pass the captured UCB address in register 1 rather than in a parameter list but does not provide input parameter validation or recovery.

For information about accessing an above 16 megabyte UCB, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	CAPTUCB or UCAPTUCB option: problem state with PSW key 0-7, or supervisor state CAPTOACT option with ASID: problem state with PSW key 0-7, or supervisor state CAPTOACT option without ASID: problem state and any PSW key. For any of the options with LINKAGE=BRANCH: supervisor state with PSW key 0 is required.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	CAPTUCB or UCAPTUCB option: PASN=HASN=SASN CAPTOACT option: any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or Access register (AR)

Environmental factor	Requirement
Interrupt status:	CAPTUCB or UCAPTUCB option with CAPTCOM=NO: enabled for I/O and external interrupts CAPTUCB option with CAPTCOM=NEVER: enabled for I/O interrupts. CAPTUCB or UCAPTUCB option with CAPTCOM=YES: enabled or disabled for I/O and external interrupts. CAPTOACT option: enabled or disabled for I/O and external interrupts.
Locks:	CAPTUCB or UCAPTUCB option with CAPTCOM=NO: no locks held. CAPTUCB option with CAPTCOM=NEVER: no locks held. CAPTUCB or UCAPTUCB option with CAPTCOM=YES: the caller may hold locks, but is not required to hold any. CAPTOACT option: the caller may hold locks, but is not required to hold any.
Control parameters:	If the caller of IOSCAPU with the CAPTOACT option is disabled, the parameter list must be in nonpageable or disabled reference (DREF) storage. This situation is also true for a caller of IOSCAPU with the CAPTUCB or UCAPTUCB option and the CAPTCOM=YES parameter.

Programming requirements

The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

Restrictions

Only use IOSCAPU CAPTOACT without ASID, to translate a captured UCB address that was captured in your primary address space.

Input register information

Before issuing the IOSCAPU macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Capture an UCB function

Syntax

The standard form of the IOSCAPU macro with the CAPTUCB option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
CAPTUCB	
,UCBPTR=*ucbptr*	*ucbptr* :RS-type or address in register (2) - (12).
,CAPTPTR=*captptr*	*captptr* :RS-type or address in register (2) - (12).
,LASTING=NO	**Default**: LASTING=NO
,LASTING=YES	
,CAPTCOM=NO	**Default**: CAPTCOM=NO
,CAPTCOM=YES	
,CAPTCOM=NEVER	
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM

Syntax	Description
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode addr*	*rsncode addr* : RS-type address or address in register (2) - (12) of fullword output variable.

Parameters

The parameters are explained as follows:

CAPTUCB
Specifies that you want to capture an actual UCB into the private storage area of an address space. Capturing the UCB enables you to access the UCB with a 24-bit address.

,UCBPTR=*ucbptr*
Specifies a pointer that contains the address of the common segment of the actual UCB that you want to capture.

,CAPTPTR=*captptr*
Specifies the pointer to contain the address of the common segment of the captured UCB.

Note: CAPTPTR is a four byte field. If the caller specifies a field with a high order byte for flags, those flags are overlaid when the macro specifies the output pointer to the captured UCB.

,LASTING=YES
,LASTING=NO
Specifies whether the system should release the captured UCB automatically during end of task termination.

- **NO**: Frees any captured UCBs during the end of job step task
- **YES**: Leaves any captured UCBs during the end of job step task

Note: If, and only if, LASTING=YES is specified when capturing a UCB, LASTING=YES should be specified when releasing the same captured UCB.

,CAPTCOM=NO
,CAPTCOM=YES
,CAPTCOM=NEVER
Specifies whether the above 16 megabyte UCB should be captured into common storage.

- **NO**: Capture the UCB into private storage of the current address space
- **YES**: Capture the UCB into common storage. This option is not recommended because it uses common storage.
- **NEVER**: Unconditionally capture the UCB into private storage of the current address space.

  Note:

  Since there are reasons why a CAPTCOM=NO request may still cause a UCB to be captured to common (i.e., If the UCB is already captured in common), this keyword can be used to force IOS to capture the UCB to private storage.

  Specifying CAPTCOM=NEVER may cause duplicate UCB storage to be allocated in the case where the UCB is already captured to common.

Since captures done in MASTERS address space will always be captured to common storage, the CAPTCOM=NEVER specification will be ignored in this case.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Specifies the type of call that should be generated:

- **SYSTEM**: Specifies a program call (PC)
- **BRANCH**: Specifies a branch entry

,RETCODE=retcode addr
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=rsncode addr
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

ABEND codes

IOSCAPU might abnormally end with abend code X'2C6'. See *z/OS MVS System Codes* for an explanation of abend code X'2C6'.

Return and reason codes

When the IOSCAPU macro returns control to your program, GPR 15 (and *retcode* if you coded RETCODE) contains the return code. If the return code is not 0, GPR0 (and *rsncode* if you coded RSNCODE) contains the reason code.

Table 91. Return and Reason Codes for the IOSCAPU CAPTUCB Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: IOSCAPU completed successfully. **Action**: None.
04	04	**Meaning**: Warning. The program attempted to capture a below 16 megabyte UCB. The address of the actual UCB is returned and a capture is not performed. **Action**: None required if the program attempts to capture any input UCB. Otherwise, check the address of the actual UCB. Correct the error and rerun the program.
08	0C	**Meaning**: Warning. The program attempted to capture a UCB that was at a captured UCB address. **Action**: None required if the program attempts to capture any input UCB. Otherwise, check to see if your program correctly specified the actual UCB address on the UCBPTR parameter. Correct the error and rerun the program.
08	10	**Meaning**: Program error. The program attempted to use a UCB address that is not a valid UCB. **Action**: Check to see if your program correctly specified the UCB address on the UCBPTR parameter. Correct the error and rerun the program.
20		**Meaning**: System error. This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain storage that it required. **Action**: Record the return code and supply it to the appropriate IBM support personnel.

Example

Capture a UCB at the address specified by UCBPTR and receive the captured UCB address in CAPTURED.

```
IOS_CAPT IOSCAPU CAPTUCB,                                              X
               UCBPTR=UCBPTR,                                          X
```

```
                CAPTPTR=CAPTURED,                                      X
                LINKAGE=BRANCH,                                        X
                MF=(E,CAPTLIST)
```

IOSCAPU CAPTUCB - List form

Use the list form of the IOSCAPU macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

Syntax

The list form of the IOSCAPU macro with the CAPTUCB option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default**: 0D

Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOSCAPU macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of X'0D', which forces the parameter list to a doubleword boundary.

IOSCAPU CAPTUCB - Execute form

Use the execute form of the IOSCAPU macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSCAPU macro with the CAPTUCB option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
CAPTUCB	
,UCBPTR=*ucbptr*	*ucbptr* :RS-type or address in register (2) - (12).
,CAPTPTR=*captptr*	*captptr* :RS-type or address in register (2) - (12).
,LASTING=NO	**Default**: LASTING=NO
,LASTING=YES	
,CAPTCOM=NO	**Default**: CAPTCOM=NO
,CAPTCOM=YES	
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode addr*	*rsncode addr* : RS-type address or address in register (2) - (12) of fullword output variable.

Syntax	Description
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
Specifies the execute form of the IOSCAPU macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

Release a captured UCB function

Syntax

The standard form of the IOSCAPU macro with the UCAPTUCB option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IOSCAPU.
IOSCAPU	
␣	One or more blanks must follow IOSCAPU.
UCAPTUCB	
,CAPTPTR=*captptr*	*captptr* :RS-type or address in register (2) - (12).
,LASTING=NO	**Default**: LASTING=NO
,LASTING=YES	

Syntax	Description
,CAPTCOM=NO	**Default**: CAPTCOM=NO
,CAPTCOM=YES	
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode addr*	*rsncode addr* : RS-type address or address in register (2) - (12) of fullword output variable.

Parameters

The parameters are explained as follows:

UCAPTUCB
Specifies that you want a captured UCB released.

,CAPTPTR=*captptr*
Specifies the address of the common segment of the captured UCB that you want released.

,CAPTCOM=NO
,CAPTCOM=YES
Specifies whether the above 16 megabyte UCB should be released from common storage.

- **NO**: Release the UCB from private storage of the current address space
- **YES**: Release the UCB from common storage.

,LASTING=NO
This is NOT a UCAPTUCB associated with a previous CAPTUCB (where LASTING=YES was specified).

,LASTING=YES
This is a UCAPTUCB associated with a previous CAPTUCB where LASTING=YES was specified).

Note: If, and only if, LASTING=YES is specified when capturing a UCB, LASTING=YES should be specified when releasing the same captured UCB.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Specifies the type of call that should be generated:

- **SYSTEM**: Specifies a program call (PC)
- **BRANCH**: Specifies a branch entry

,RETCODE=retcode addr
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=rsncode addr
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

ABEND codes

IOSCAPU might abnormally end with abend code X'2C6'. See *z/OS MVS System Codes* for an explanation of abend code X'2C6'.

Return and reason codes

When the IOSCAPU macro returns control to your program, GPR 15 (and *retcode* if you coded RETCODE) contains the return code. If the return code is not 0, GPR0 (and *rsncode* if you coded RSNCODE) contains the reason code.

Table 92. Return and Reason Codes for the IOSCAPU UCAPTUCB Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: IOSCAPU completed successfully. **Action**: None.
04	08	**Meaning**: Warning. The program attempted to release an actual below 16 megabyte UCB. **Action**: None required if the program tries to release any input UCB. Otherwise, check the address of the captured UCB. Correct the error and rerun the program.
08	08	**Meaning**: Program error. The program attempted to release a captured UCB and the captured UCB does not exist in the address space. **Action**: Check to see if your program correctly specified the captured UCB address on the CAPTPTR parameter. Correct the error and rerun the program.
08	10	**Meaning**: Program error. The program attempted to use a UCB address that is not a valid UCB. **Action**: Check to see if your program correctly specified the UCB address on the UCBPTR or CAPTPTR parameter. Correct the error and rerun the program.
08	18	**Meaning**: Warning. The program attempted to release an actual above 16 megabyte UCB. **Action**: None required if the program tries to release any input UCB. Otherwise, check the address of the captured UCB. Correct the error and rerun the program.
20		**Meaning**: System error. This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain storage that it required. **Action**: Record the return code and supply it to the appropriate IBM support personnel.

Example

Release the captured UCB at the address specified by CAPTURED.

```
IOS_UNCA IOSCAPU UCAPTUCB,                                          X
               CAPTPTR=CAPTURED,                                    X
               LINKAGE=BRANCH,                                      X
               MF=(E,CAPTLIST)
```

IOSCAPU UCAPTUCB - List form

Use the list form of the IOSCAPU macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

Syntax

The list form of the IOSCAPU macro with the UCAPTUCB option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default**: 0D

Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOSCAPU macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of X'0D', which forces the parameter list to a doubleword boundary.

IOSCAPU UCAPTUCB - Execute form

Use the execute form of the IOSCAPU macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSCAPU macro with the UCAPTUCB option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.

Syntax	Description
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
UCAPTUCB	
,CAPTPTR=*captptr*	*captptr* :RS-type or address in register (2) - (12).
,CAPTCOM=NO	**Default**: CAPTCOM=NO
,CAPTCOM=YES	
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode addr*	*rsncode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
Specifies the execute form of the IOSCAPU macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

Translate captured to actual address function

Syntax

The standard form of the IOSCAPU macro with the CAPTOACT option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
CAPTOACT	
,CAPTPTR=*captptr*	*captptr* :RS-type or address in register (2) - (12).
,UCBPTR=*ucbptr*	*ucbptr* :RS-type or address in register (2) - (12).
,ASID=CURRENT	**Default**: ASID=CURRENT
,ASID=*asid*	*asid* :RS-type or address in register (2) - (12).
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode addr*	*rsncode addr* : RS-type address or address in register (2) - (12) of fullword output variable.

Parameters

The parameters are explained as follows:

CAPTOACT
Specifies that you want the actual UCB address for a captured UCB.

,CAPTPTR=*captptr*
Specifies the pointer to the address of the common segment of the captured UCB.

,UCBPTR=*ucbptr*
Specifies a pointer to contain the address of the actual UCB common segment.

,ASID=CURRENT
,ASID=*asid*
Specifies the address space in which the captured UCB was originally captured.

- **CURRENT**: Specifies the address space of the program

- **asid**: Specifies the name of another address space

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Specifies the type of call that should be generated:

- **SYSTEM**: Specifies a program call (PC)
- **BRANCH**: Specifies a branch entry

,RETCODE=retcode addr
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=rsncode addr
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0.

ABEND codes

IOSCAPU might abnormally end with abend code X'2C6'. See *z/OS MVS System Codes* for an explanation of abend code X'2C6'.

Return and reason codes

When the IOSCAPU macro returns control to your program, GPR 15 (and *retcode* if you coded RETCODE) contains the return code. If the return code is not 0, GPR0 (and *rsncode* if you coded RSNCODE) contains the reason code.

Table 93. Return and Reason Codes for the IOSCAPU CAPTOACT Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: IOSCAPU completed successfully. **Action**: None.
04	0C	**Meaning**: Warning. The program requested the actual address of an actual UCB. **Action**: None required if the program always attempts to receive the actual UCB address for a UCB. Otherwise, check the address of the captured UCB. Correct the error and rerun the program.
08	04	**Meaning**: Program error. The program attempted to receive the actual UCB address for a captured UCB and the address space identifier specified for the captured UCB does not exist or the address space was swapped out. **Action**: Retry the request because the address space might have been swapped in. Also, check to see if your program correctly specified the address space of the captured UCB on the ASID parameter. Correct the error and rerun the program.
08	10	**Meaning**: Program error. The program attempted to use a UCB address that is not a valid UCB. **Action**: Check to see if your program correctly specified the UCB address on the CAPTPTR parameter. Correct the error and rerun the program.
20		**Meaning**: System error. This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain storage that it required. **Action**: Record the return code and supply it to the appropriate IBM support personnel.

Example

Receive the actual UCB address (in ACTUAL) of the captured UCB address specified by CAPTURED.

```
IOS_TRAN IOSCAPU CAPTOACT,                                      X
               UCBPTR=ACTUAL,                                   X
```

```
                CAPTPTR=CAPTURED,                                      X
                LINKAGE=BRANCH,                                        X
                MF=(E,CAPTLIST)
```

IOSCAPU CAPTOACT - List form

Use the list form of the IOSCAPU macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

Syntax

The list form of the IOSCAPU macro with the CAPTOACT option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default**: 0D

Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOSCAPU macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of X'0D', which forces the parameter list to a doubleword boundary.

IOSCAPU CAPTOACT - Execute form

Use the execute form of the IOSCAPU macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSCAPU macro with the CAPTOACT option is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCAPU.
IOSCAPU	
ƀ	One or more blanks must follow IOSCAPU.
CAPTOACT	
,UCBPTR=*ucbptr*	*ucbptr* :RS-type or address in register (2) - (12).
,CAPTPTR=*captptr*	*captptr* :RS-type or address in register (2) - (12).
,ASID=CURRENT	**Default**: ASID=CURRENT
,ASID=*asid*	*asid* :RS-type or address in register (2) - (12).
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode addr*	*retcode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,RSNCODE=*rsncode addr*	*rsncode addr* : RS-type address or address in register (2) - (12) of fullword output variable.
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Syntax	Description

Parameters

The parameters are explained under the standard form of the IOSCAPU macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)

Specifies the execute form of the IOSCAPU macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

Chapter 99. IOSCDR — Retrieve configuration data records

Description

The IOSCDR macro enables authorized callers to retrieve device identification information (such as the serial number and the model number) for an I/O device located along a specific I/O path. This information can allow installation management to do the following:

- Uniquely identify, across multiple systems, I/O hardware located along a specific I/O path
- Following device installs, check device paths to ensure that cables are connected to the proper device before bringing the device or path online
- Construct a map of an installation's configuration
- During problem diagnosis, ensure that all paths to a given device are reaching the expected device.

The configuration data record (CDR) information that IOSCDR retrieves is mapped by the mapping macro IHACDR.

The format of IHACDR is in *z/OS MVS Data Areas, Vol 3 (IEFDORC-ISGYQCBP)*. For more information about the contents of CDRs and information about the contents of node descriptors (NDs), see *ESA/390 Common I/O Device Commands*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum Authorization:	For LINKAGE=LINK, supervisor state and any PSW key. For LINKAGE=SYSTEM, any one or more of the following: • Supervisor state • PKM allowing key 0 – 7 • PSW key 0 – 7 • APF-authorized • RACF authorization to the FACILITY class and the IOSCDR entity
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

Include the IHACDR mapping macro.

Restrictions

The caller can have no enabled, unlocked task (EUT) FRRs established.

Note that, when you issue IOSCDR, the service pins the device so that the device's UCB and other related data structures are not dynamically deleted while IOSCDR is retrieving the data. When IOSCDR completes, it unpins the device.

Input register information

Before issuing the IOSCDR macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller of the IOSCDR macro, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code

1
Used as a work register by IOSCDR

2-13
Unchanged

14
Used as a work register by IOSCDR

15
Return code

When control returns to the caller of the IOSCDR macro, the access registers (ARs) contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IOSCDR macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCDR.

Syntax	Description
IOSCDR	
ƀ	One or more blanks must follow IOSCDR.
DEVN=*device num*	*device num:* RX-type address or address in register (2) - (12).
,SCHSET=*xschset*	*xschset:* RX-type address or register (2) - (12).
,CHPID=*path id*	*path id:* RX-type address or address in register (2) - (12).
,CDRAREA=*cdr area*	*cdr area*: RX-type address or address in register (2) - (12).
,CDRLEN=*cdr length*	*cdr length*: RX-type address or address in register (2) - (12).
,CDRSIZE=*cdr size*	*cdr size*: RX-type address or address in register (2) - (12).
,LINKAGE=SYSTEM	**Default**: LINK
,LINKAGE=LINK	
,NODE_DESCRIPTOR=*node descriptor*	Optional input. It is the name (RS-type), or address in register (2)-(12), of the 32 bytes of storage for one node descriptor to be returned by the service. The *node descriptor* is associated with the control unit that is attached to the specified path in the input mask.
,READ=NOIO	
,READ=IO	
,READ=COND	
,STATUS=*status*	*status*: RX-type address or address in register (2) - (12).
,TIME=*time*	*time*: RX-type address or address in register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or address in register (2) - (12).
,WWPN=*xwwpn*	*xwwpn*: RX-type address or address in register (2) - (12).
,RETCODE=*return code*	*return code*: RX-type address or address in register (2) - (12).
,RSNCODE=*reason code*	*reason code*: RX-type address or address in register (2) - (12).

Syntax	Description
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	
,PLISTVER=*parameter list version*	

Parameter descriptions

The parameters are explained as follows:

,DEVN=*device num*
Specifies the binary device number (0000 - FFFF) of a device for which IOSCDR retrieves a CDR.

,SCHSET=*xschset*
,SCHSET=*0*
Specifies the name (RS-type), or address in register (2)-(12), of an optional byte input that specifies a subchannel set for the CDR that is to be retrieved. DEFAULT: 0.

,CHPID=*path id*
Specifies the channel path ID (00 - FF) of a specific path for which IOSCDR retrieves a CDR. To determine the ID for a specific channel path, use the UCBINFO PATHINFO macro or the DISPLAY MATRIX operator command.

,CDRAREA=*cdr area*
Specifies the name of the work area that receives a copy of the CDR for the specified device and path. You must specify on the CDRLEN parameter the length of the CDR area. The CDR area is mapped by IHACDR. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for more information about IHACDR.

,CDRLEN=*cdr length*
Specifies the length of the CDR area. The maximum length of the CDR area is 65535 bytes. You can start with a length of 256 bytes. If the length you specify is smaller than the CDR, IOSCDR returns only a partial CDR and the caller receives return code X'04' and reason code X'04'. To ensure that IOSCDR returned the entire CDR, verify that the value returned on CDRSIZE is less than or equal to CDRLEN. Note that CDRSIZE is device dependent.

,CDRSIZE=*cdr size*
Specifies the area into which IOSCDR returns the actual size of the CDR for the specified device and path. You can use this parameter for diagnostic purposes to determine how large CDRLEN should be.

,LINKAGE=SYSTEM
,LINKAGE=LINK
Specifies the type of call that should be generated:

- **SYSTEM**: Specifies a Program Call (PC) that passes control to the service routine. The caller does not have to be in supervisor state.
- **LINK**: Specifies a LINK macro call to pass control to the service routine. This call is more direct but the caller must be in supervisor state.

,NODE_DESCRIPTOR=*node descriptor area*
Specifies the name of the work area that receives a copy of the node descriptor for the specified device and path.

,READ=NOIO
,READ=IO
,READ=COND
Specifies that IOSCDR retrieves the CDR or ND in one of the following ways:

- NOIO—IOSCDR retrieves the last CDR or ND known to MVS for the specified device and path. Note that this CDR or ND might not exist or might not be current if the specified device is offline. If the specified device is online, this option is fairly reliable and is quick because no I/O is performed.
- IO—IOSCDR retrieves the CDR or ND for a specified device and path directly from the specified device. Although this option is slower than READ=NOIO, READ=IO provides more current information.
- COND— If the specified device and path are online, IOSCDR retrieves the last CDR or ND known to MVS for the specified device and path. Otherwise, IOSCDR retrieves the CDR directly from the device. COND is the best option to choose if you are interested in retrieving the most accurate CDR or ND in the shortest time possible. Note that the READ parameter has no effect on the way in which a WWPN is retrieved; only the last WWPN known to MVS can be returned.

,STATUS=*status*

Specifies a one-byte field containing status information about successful invocations of IOSCDR. The bit positions, represented in hexadecimal values, are as follows:

Bit	Status	Meaning
0	on off	CDR returned was read from the device. CDR returned was the last CDR known to MVS.
1	on off	Specified CHPID was logically online to the device. Specified CHPID was logically offline to the device.
2	on off	Specified device was online. Specified device was offline.
3 - 7	—	Reserved for IBM use.

,TIME=*time*

Specifies an 8-byte field containing the maximum amount of time, in seconds, that IOSCDR can run before being purged. The default for the TIME parameter is 5 seconds. You can use TIME when you specify READ=COND or READ=IO. When you specify READ=NOIO, IOSCDR ignores the TIME parameter.

The time interval, whose address resides in virtual storage, is presented as zoned decimal digits in the form:

HHMMSSth, where:

HH
is hours (24-hour clock)

MM
is minutes

SS
is seconds

t
is tenths of seconds

h
is hundredths of seconds

IOSCDR runs until one of the following occurs:

- IOSCDR completes successfully or unsuccessfully
- The interval that you specify on the TIME=parameter expires
- The MIH interval for the device expires.

Note that the TIME parameter allows you to set an expiration time that is specific to IOSCDR. The MIH interval, however, is used by other services associated with the device. Using the TIME parameter allows you to set an expiration time that is shorter than the MIH interval.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character area that contains the MVS I/O configuration token that you supply to IOSCDR. You can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when IOSCDR is invoked does not match the token whose address you supply as input by *ioctoken addr*, you receive an error return code.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, IOSCDR sets IOCTOKEN to the current I/O configuration token.

For information about how you can use the configuration token to detect configuration changes, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

,WWPN=*xwwpn*
Specifies the location where the system is to place the Worldwide Port Name (WWPN) for the port on the control unit for the specified channel path. If the WWPN is not available, zeroes will be returned.

,RETCODE=*return code*
Specifies the location or register where the system is to place the return code. The system copies the return code into the location from register 15.

,RSNCODE=*reason code*
Specifies the location or register where the system is to place the reason code. The system copies the reason code into the location from register 0.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*parameter list version*
A decimal value in the "1-2" range that specifies the macro version. PLISTVER determines which parameter list is generated. Note that MAX can be specified instead of a number, and the parameter list will be the largest size currently supported. This size might grow from release to release, thus possibly affecting the amount of storage needed by your program. If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form of the parameter list, as that will ensure that the list form parameter list is always long enough to hold whatever parameters that might be specified on the execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version that allows all of the parameters specified on the invocation to be processed.

Return codes

Return and reason codes, in hexadecimal, from the IOSCDR macro are as follows:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: IOSCDR processing completed successfully. IOSCDR successfully retrieved the CDR for the specified device and path. **Action**: None
04	04	**Meaning**: IOSCDR cannot retrieve an entire CDR because the CDR area specified was not large enough to receive the CDR. **Action**: The size of the CDR area is determined by CDRLEN. If you do not know what length to specify on CDRLEN, use the optional CDRSIZE parameter. If you specified CDRSIZE, IOSCDR returns the size that CDRAREA needs to be. Retry the operation with a CDR area of the same length as the length returned on CDRSIZE for the failing operation.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
04	08	**Meaning**: IOSCDR cannot retrieve the CDR for the specified device and path. If you specified READ=IO, either a subchannel error or an I/O error could be preventing IOSCDR from retrieving the CDR. If you specified READ=NOIO, a subchannel error could be preventing IOSCDR from retrieving the CDR. **Action**: Further investigation of the problem is required. The D M=DEV command may provide further diagnosis data. For example, a subchannel error may have occurred because the device is not available. Attempt to vary the path online to produce further diagnosis data. See *ESCON Error Recovery Concepts and Procedures in an MVS Environment* for further problem diagnosis information. If the problem persists, contact your IBM service representative.
04	0C	**Meaning**: IOSCDR cannot retrieve the CDR for the specified device and path. I/O was attempted to the device, but the time interval specified on the TIME parameter expired before I/O completed. **Action**: Verify that the time interval was sufficiently long. Note that the system issues this return code only if the time expired before the device's MIH interval. To determine the MIH interval, use the 'D MIH' command or the MIHQUERY macro.
04	10	**Meaning**: IOSCDR cannot retrieve the CDR for the specified device and path because MVS does not have a last known CDR to return. **Action**: Use one of the following methods to retrieve a CDR: • Bring the device and path online. If a CDR is available, the system will store it. • Retrieve the CDR directly from the device, by issuing the IOSCDR macro with the READ=IO option.
04	14	**Meaning**: IOSCDR cannot retrieve the last known CDR. IOSCDR did not attempt I/O. **Action**: A system problem exists that prevents any last known CDR from being retrieved. Retry the operation. If the problem persists, contact IBM Software Support.
08	04	**Meaning**: The specified device does not support the channel control words (CCWs) used to obtain configuration data records. **Action**: None
08	08	**Meaning**: IOSCDR cannot retrieve the CDR because the device number specified on the DEVN parameter is not valid. **Action**: Verify your program to ensure that the correct device was passed and retry the operation. If the device number is valid, use the IOCTOKEN keyword to ensure that the device is not dynamically changed or deleted.
08	0C	**Meaning**: IOSCDR cannot retrieve the CDR because the channel path id on the CHPID parameter is not valid. **Action**: Verify your program to ensure that the correct CHPID was passed and retry the operation. Use the IOCTOKEN keyword to ensure that the CHPID for the device was not dynamically changed or deleted.
08	10	**Meaning**: IOSCDR cannot retrieve the CDR because the time specified on the TIME keyword is not valid. **Action**: Ensure that the time specified contains valid zoned decimal digits that are in the proper range.
08	14	**Meaning**: An incorrect CDR length was specified on the CDRLEN keyword. **Action**: Verify that CDRLEN is greater than 0 and does not exceed 65535 bytes, then retry the operation.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	20	**Meaning**: IOSCDR cannot retrieve the CDR because the I/O configuration token that is current when IOSCDR is invoked does not match the token whose address is supplied as input by IOCTOKEN. Note that this return code is only valid for callers using the IOCTOKEN keyword. **Action**: Ensure that the device number and CHPID are still valid and retry the operation passing a current IOCTOKEN.
08	24	**Meaning**: IOSCDR cannot retrieve the CDR because the IOS address space is not yet available. **Action**: Retry the operation after the IOS address space is available (master scheduler initialization has completed).
08	28	**Meaning**: IOSCDR cannot establish an ESTAE. **Action**: Ensure that there is sufficient private area storage, then retry the operation.
08	2Cx	**Meaning**: The value specified on the SCHSET keyword is not valid. **Action**: Supply the correct value on the SCHSET keyword.
0C	None	**Meaning**: An unexpected error occurred. **Action**: Record the return code and supply it to the appropriate IBM support personnel.

Example

Assume you want to retrieve a configuration data record (CDR) to determine if the manufacturer of a SYSRES volume is IBM.

Scan through all UCBs using the UCBSCAN macro, and put copies of the DASD UCBs the program finds in a user-supplied work area called UCBSTOR. When the program finds the SYSRES device, issue the UCBINFO macro to obtain information about the device path and type of channel path for the specified UCB. Information, such as the channel path ID and online status, will appear in the IOSDPATH data area. The program looks through the channel path information until it finds an online path, then issues the IOSCDR macro to retrieve the CDR containing information about the manufacturer of the SYSRES volume.

```
*.....................................................................*
*         REGISTER ASSIGNMENTS                                        *
*.....................................................................*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6                    Dynamic area register
UCBPTR7  EQU   7                    UCB Pointer
R8       EQU   8
R9       EQU   9                    Module base register
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13                   Pointer to standard save area
R14      EQU   14
R15      EQU   15
         SPACE 3
         TITLE 'IOSSCDRE -  IOSCDR Sample Program'
*.....................................................................*
*                                                                     *
*        Standard Entry Linkage                                       *
*                                                                     *
*.....................................................................*
         PRINT GEN
         USING *,R9              Sets up base register
ENTRY    STM   R14,R12,12(R13)   Save caller's registers
         LR    R9,R15            Establish module base register
         MODESET KEY=ZERO,MODE=SUP
         LA    R0,DYNSIZE        Load length of dynamic area
```

```
         STORAGE OBTAIN,LENGTH=((R0)),SP=233 Gets dynamic area
         LR    R6,R1              Gets dynamic area address
         USING DYNAREA,R6         Sets up dynamic area
         ST    R13,SAVE+4         Save caller's save area address
         LA    R15,SAVE           Get this module's save area address
         ST    R15,8(R13)         Save this modules save area address
*                                 in caller's save area.
         LR    R13,R15            Set up addressability to this
*                                 module's save area.
         B     MAINLINE
         DC    CL8'IOSSCDRE'
         DC    CL8'&SYSDATE'
         DC    CL8'&SYSTIME'
         TITLE 'IOSSCDRE - SCDRE mainline '
*.....................................................................*
*                                                                     *
*        MAINLINE                                                     *
*                                                                     *
*.....................................................................*
MAINLINE DS   0H
*
```

```
              L     10,X'10'           Load CVT pointer
              USING CVT,10
              TM   CVTDCB,CVTOSEXT     Is the OSLEVEL extension present
              BNO  NO_IOSCDR           No, pre-MVS/SP Version 3 system
*
              TM   CVTOSLV1,CVTH5510   Running on version HBB5510?
              BNO  NO_IOSCDR           No, pre-HBB5510 system.  IOSCDR
*                                      supported on HBB5510 and above
*.....................................................................*
*                                                                     *
*   Set up addressability to a storage area called UCBSTOR into which *
*   the UCBSCAN macro will return the UCBs of devices it locates.     *
*                                                                     *
*.....................................................................*
              LA       UCBPTR7,UCBSTOR    Get address of work area
              USING    UCB,UCBPTR7        Set up addressability
*
*.....................................................................*
*                                                                     *
*   Clear the UCBSCAN work area.                                      *
*                                                                     *
*.....................................................................*
              LA  R0,SCANWORK             Set storage address
              LA  R1,100                  Set storage length
              SR  R15,R15                 Clear second operand
              MVCL  R0,R14                Clear the storage
*.....................................................................*
*                                                                     *
*   Loop through all DASD UCBs looking for the SYSRES volume.         *
*                                                                     *
*   Note: There must be a SYSRES volume, and hence it will be found   *
*         in the scan loop which follows.                             *
*                                                                     *
*.....................................................................*
SCANLOOP UCBSCAN COPY,                                                X
               WORKAREA=SCANWORK,                                     X
               UCBAREA=UCBSTOR,                                       X
               DEVCLASS=DASD,                                         X
               MF=(E,SCANLIST)
*.....................................................................*
*                                                                     *
*   If UCBSCAN returned a UCB, check whether it is the SYSRES         *
*   volume.  If it isn't, continue checking more UCBs.  If            *
*   the UCB represents the SYSRES device, end the loop.               *
*                                                                     *
*.....................................................................*
              LTR R15,R15            Test return code
              BNZ EXIT_ERROR         Exit if non-zero
              TM UCBSTAT,UCBSYSR     Test if SYSRES volume
              BZ SCANLOOP            Keep looping if not
*
```

```
*.....................................................................*
*                                                                     *
*    Issue the UCBINFO macro to obtain path-related information.      *
*    UCBINFO returns this information in a field called PATHSTOR,     *
*    mapped by IOSDPATH.                                              *
*                                                                     *
```

```
*    Note- Since the device whose path information is sought is the   *
*          SYSRES device, an online path is certain to be found.      *
*          No loop counter is used.                                   *
*.....................................................................*
*
         UCBINFO PATHINFO,                                            X
               DEVN=UCBCHAN,                                          X
               PATHAREA=PATHSTOR,                                     X
               MF=(E,INFOLIST)
*.....................................................................*
*                                                                     *
*   If UCBINFO cannot retrieve path-related information, that is, you *
*   receive a non-zero return code, exit program.                     *
*                                                                     *
*.....................................................................*
              LTR      R15,R15           Test for 0 return code
              BNZ      EXIT_ERROR        Exit if bad RC
*.....................................................................*
*                                                                     *
*   Loop through the channel path ID array entries returned in        *
*   PATHSTOR to find the first online path.  An online path           *
*   is represented by a flag in the array.                            *
*                                                                     *
*.....................................................................*
              LA       R10,PATHSTOR      Address of PATHINFO data
              USING    PATH,R10          Set up addressability to
*                                        path information.
              SR       R8,R8             CHPID array index register.
CHPID_LOOP    IC       R11,PATHBITS(R8)  Get flags from array entry.
              STC      R11,PATHSAVE      Save entry
              TM       PATHSAVE,X'04'    Test if the path is online
              BO       CHPID_EXIT        If so, exit the loop
              LA       R8,L'PATHCHPIDARRAY(R8)   Increment array index
              B        CHPID_LOOP
CHPID_EXIT    LH       R11,PATHCHPID(R8) Get the ID for the online
*                                        channel path.
              STC      R11,CHPID         Save the ID for the online
*                                        channel path.
*.....................................................................*
*                                                                     *
*   The program identifies an online channel path to the SYSRES       *
*   volume.                                                           *
*   Issue the IOSCDR macro to request a configuration data            *
*   record (CDR) for the SYSRES volume whose binary number            *
*   you specify in the UCBCHAN field.  IOSCDR returns the CDR         *
*   in a storage area called CDRSTOR, whose length you specify        *
*   on the CDRLEN parameter.                                          *
*   Specify the channel path ID (CHPID) of the online                 *
*   path returned by the UCBINFO macro.  Also specify                 *
*   the IOSCDR READ=NOIO option to avoid performing                   *
*   I/O operations to the SYSRES volume.  The IOSCDR READ=NOIO        *
*   option will have a CDR to return if the device                    *
*   supports the self-description channel control words (CCWs).       *
*                                                                     *
*.....................................................................*
```

```
         IOSCDR DEVN=UCBCHAN,                                         X
               CHPID=CHPID,                                           X
               READ=NOIO,                                             X
               CDRAREA=CDRSTOR,                                       X
               CDRLEN=CDRLEN,                                         X
               CDRSIZE=CDRSIZE,                                       X
               MF=(E,CDRLIST)
*.....................................................................*
*                                                                     *
*   Check for a zero return code, indicating that IOSCDR completed    *
*   successfully.  If it was not successful, examine the return       *
*   and reason codes to determine the cause.                          *
*                                                                     *
*   Note:  A large CDRAREA was specified for the purposes of this     *
*          example to reduce the possibility of the CDRAREA being     *
*          too small to contain the returned CDR.  It                 *
*          is expected that in practical applications of the IOSCDR   *
*          service, users will obtain the CDRAREA by issuing the      *
*          GETMAIN macro.  If the IOSCDR macro indicates              *
*          through return and reason codes that the                   *
*          area passed was too small, issue the FREEMAIN macro to     *
*          release the storage, and obtain a larger area.  Reissue    *
*          the IOSCDR macro.  IOSCDR indicates the minimum size       *
*          for the CDRAREA through the CDRSIZE keyword.               *
*                                                                     *
```

```
*.....................................................................*
             LTR      R15,R15          Test for 0 return code
             BNZ      EXIT_ERROR       Exit if bad RC
*.....................................................................*
*                                                                     *
*   Scan the CDR, mapped by IHACDR, searching for the node element    *
*   descriptor (NED) for the SYSRES volume.  The NEDTCU field         *
*   should indicate that this device is a control unit.               *
*                                                                     *
*.....................................................................*
             LA  R10,CDRSTOR      Set up addressability to the
*                                 CDRAREA.
             USING NED,R10
             SR  R8,R8            Clear NED index register.
CDR_LOOP     TM  NEDFLAGS,CDRFNED Check if the record represents an
*                                 NED.
             BNO CDR_ITERATE      If not, try next record.
             CLI NEDTYPE,NEDTCU   Check if the NED represents a
*                                 control unit.
             BNE CDR_ITERATE      If not, try next record.
             B   CDR_EXIT         CU NED found.
CDR_ITERATE  LA  R8,32(R8)        Increment index register.
             LA  R10,32(R10)      Increment to next record in CDR.
             CL  R8,CDRSIZE       Make sure that there are more
*                                 records.
             BL  CDR_LOOP         Iterate loop.
             B EXIT_ERROR         No CU NED found.  Exit program
*.....................................................................*
*                                                                     *
*   If the program finds the NED, check if IBM manufactured           *
*   the control unit by looking in the NEDMANUF field of the          *
*   returned CDR. Check if the control unit was manufactured          *
*   by IBM.  Return a WTO to the user describing the result.          *
*                                                                     *
*.....................................................................*
```

```
CDR_EXIT     DS 0D
             CLC NEDMANUF,=CL3'IBM'   Check if built by IBM
             BNE NOT_IBM
             B IS_IBM
IS_IBM       DS 0D
         WTO   'IOSSCDRE-CONTROL UNIT FOR SYSRES WAS BUILT BY IBM',   X
               ROUTCDE=(11),DESC=(2)
             B EXIT
NOT_IBM      DS 0D
         WTO  'IOSSCDRE-CONTROL UNIT FOR SYSRES WAS NOT BUILT BY IBM',X
               ROUTCDE=(11),DESC=(2)
*
             B EXIT
*.....................................................................*
*                                                                     *
*    Return a WTO to the user saying that the IOSCDR macro            *
*    is not available on the system executing this sample program.    *
*                                                                     *
*.....................................................................*
NO_IOSCDR    DS       0H
         WTO   'IOSSCDRE - IOSCDR SUPPORTED IN HBB5510 AND HIGHER',    X
               ROUTCDE=(11),DESC=(2)
             B EXIT
*.....................................................................*
*                                                                     *
*    Return a WTO to the user saying that the IOSCDR macro            *
*    encountered an error during execution of this sample program     *
*                                                                     *
*.....................................................................*
EXIT_ERROR   DS       0H
         WTO   'IOSSCDRE - THE SAMPLE ENCOUNTERED AN ERROR',           X
               ROUTCDE=(11),DESC=(2)
*.....................................................................*
*                                                                     *
*    Clean up and exit.                                               *
*                                                                     *
*.....................................................................*
EXIT           DS    0H
               L     R13,SAVE+4             Reloads caller's save
*                                           area addr into 11
               LA    R0,DYNSIZE             Loads dynamic area size
               STORAGE RELEASE,SP=233,ADDR=(R6),LENGTH=(R0)
               MODESET KEY=NZERO,MODE=PROB
               LM    R14,R12,12(R13)        Loads return regs
               BR    R14                    Returns to caller
```

```
*
*
*.....................................................................*
*                                                                     *
*    Define constants                                                 *
*                                                                     *
*.....................................................................*
CDRLEN        DC    F'512'
*.....................................................................*
*                                                                     *
*    DSECTs to map save areas and dynamic area                        *
*                                                                     *
*.....................................................................*
DYNSTART      DS   0H
DYNAREA       DSECT
*       Save area
SAVE          DS   18F
              DS   0D             Force doubleword alignment
              SPACE 2
```

```
*.....................................................................*
*                                                                     *
*  Issue the list forms of macros since the module is reentrant.      *
*                                                                     *
*.....................................................................*
LIST_INFOSERV UCBINFO MF=(L,INFOLIST)   List form of UCBINFO
INFOSERV_END  DS   0D
PATHSTOR      DS   CL256                Storage for the PATHAREA
PATHSTOR_END  DS   0D
LIST_CDRSERV  IOSCDR  MF=(L,CDRLIST)    List form of IOSCDR
CDRSERV_END   DS   0D
CDRSTOR       DS   CL512                Storage for the CDRAREA
CDRSTOR_END   DS   0D
LIST_SCANSERV UCBSCAN MF=(L,SCANLIST)   List form of UCBSCAN
SCANSERV_END  DS 0D
SCANWORK      DS CL100                  Scan work area
SCANWORK_END  DS 0D
UCBSTOR       DS CL48                   UCB copy storage
UCBSTOR_END   DS 0D
*.....................................................................*
*                                                                     *
*  Work variables and data structures local to this module            *
*                                                                     *
*.....................................................................*
CDRSIZE       DS F                    Actual size of CDR
CHPID         DS C                    CHPID used for IOSCDR invocation
PATHSAVE      DS C                    Work variable for CHPID array
*                                     entries in the PATHAREA.
END_DYN       DS    0D
DYNSIZE       EQU   *-DYNAREA         Calculates Dynamic area
*
*.....................................................................*
*                                                                     *
*  DSECTs                                                             *
*                                                                     *
*.....................................................................*
IOSSCDRE      CSECT
              TITLE 'IOSSCDRE - DSECT MAPPINGS'
         EJECT
         CVT   LIST=YES,DSECT=YES
*
UCB      DSECT
         IEFUCBOB
*
CDRAREA  IHACDR DSECT=YES
*
PATHAREA IOSDPATH
         END   IOSSCDRE
```

Chapter 100. IOSCHPD — IOS CHPID description service

Description

The IOSCHPD macro returns the acronym, description, attributes, and/or the Worldwide Port Name (WWPN) of a channel path (CHP) or channel path type.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or Supervisor state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR).
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	No locks may be held.
Control parameters:	Must be in the primary address space or be in an address/data space that is addressable through a public entry on the callers dispatchable unit access list (DU-AL).

Programming requirements

None.

Restrictions

The parameter list must be in the caller's primary address space or be addressable via the dispatchable unit access list.

The LINKAGE=BRANCH option is limited to callers which meet the following criteria:

- Supervisor state and key 0
- 31-bit addressing mode
- Primary ASC mode
- Parameter list resides in fixed or DREF storage

Input register information

Before issuing the IOSCHPD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Unpredictable (Used as a work register by the system)

2-13
Unchanged

14
Unpredictable (Used as a work register by the system)

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Unpredictable (Used as work registers by the system)

2-13
Unchanged

14-15
Unpredictable (Used as work registers by the system)

Performance implications

None.

Syntax

The IOSCHPD macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede IOSCHPD.
IOSCHPD	
␣	One or more blanks must follow IOSCHPD.
CHPID=*chpid*	*chpid*: RS-type address or register (2) - (12).
,CHP_TYPE=*chp_type*	*chp_type*: RS-type address or register (2) - (12).
,CHP_PARM=*chp_parm*	*chp_type*: RS-type address or register (2) - (12).
,CHP_PARM=0	**Default**: 0

Syntax	Description
,ACRONYM=*acronym*	*acronym*: RS-type address or register (2) - (12).
,DESC=*desc*	*desc*: RS-type address or register (2) - (12).
,ATTR=*attr*	*attr*: RS-type address or register (2) - (12).
,WWPN=*wwpn*	*wwpn*: RS-type address or register (2) - (12).
,ND=*xnd*	*xnd*: Optional 32-character output.
,LINKAGE=SYSTEM	**Default**: LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,PLISTVER=2	
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	*list addr*: RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Note: To use the IOSCHPD macro, you need to specify the following parameters:

- Either CHPID or CHP_TYPE
- One or more parameters among ACRONYM, DESC, ATTR, and WWPN

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSCHPD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

CHPID=*chpid*
An input parameter which specifies the CHPID number for which to retrieve the attributes, acronym, description, and/or WWPN. This parameter is mutually exclusive with the CHP_TYPE parameter.

If the CHPID is defined as a managed channel path, the description and acronym returned will indicate that the channel path is managed. Otherwise, a non-managed description and acronym will be returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field that contains a binary value in the range X'0000' to X'00FF'.

CHP_TYPE=*chp_type*
An input parameter which specifies the channel path type for which to retrieve the attributes, acronym, description, and/or WWPN. The channel path type can be obtained by invoking the UCBINFO PATHINFO macro and mapping the results with the IOSDPATH mapping macro. (The interface type is in the field called PathIntType). This parameter is mutually exclusive with the CHPID parameter.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

CHP_PARM=*chp_parm*
CHP_PARM=0
An optional input parameter, used only with CHP_TYPE=*chp_type* parameter, that specifies the channel path parameter. A value of 1 is the managed option and 0 (the default) is the non-managed option. If 1 is specified, and if the CHP type is managed, the description and acronym returned will indicate that the CHP type is managed.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

ATTR=*attr*
An optional parameter, used only with the CHPID parameter, that designates the output area that is to receive the CHPID attributes. The attributes are mapped by mapping macro IOSDCHPD.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,ACRONYM=*acronym*
An optional parameter that designates the output area that is to receive the acronym.

To code: Specify the RS-type address, or address in register (2)-(12), of a 5-character field.

,DESC=*desc*
An optional parameter that designates the output area that is to receive the description.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,WWPN=*wwpn*
An optional parameter, used only with the CHPID parameter, that designates the output area that is to receive the Worldwide Port Name (WWPN). (If the WWPN is not available, zeros will be returned.)

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,ND=*xnd*
An optional parameter that designates the output area that is to receive the node descriptor for the channel.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
An optional parameter that indicates whether a branch-entry linkage should be generated or a Program Call should be issued for the routine invocation. The default is LINKAGE=SYSTEM.

,LINKAGE=SYSTEM
requests Program Call invocation.

,LINKAGE=BRANCH
requests branch-entry invocation. The LINKAGE=BRANCH option is intended for performance-sensitive invokers or programs that require this function during NIP before a PC can be issued. See RESTRICTIONS for the restrictions on branch-entry invocation.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
,PLISTVER=2
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, which supports all parameters except those specifically referenced in higher versions.
- **2**, which supports ATTR and WWPN, in addition to those from version 1.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1 or 2

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the

parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of `0F` to force the parameter list to a word boundary, or `0D` to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of `0D`.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IOSCHPD macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) reason code.

The following table identifies the hexadecimal return and reason codes:

Table 94. Return and reason codes for the IOSCHPD macro

Hexadecimal return code	Reason codes, meaning, and action
00	The requested information has been returned.

Table 94. Return and reason codes for the IOSCHPD macro (continued)

Hexadecimal return code	Reason codes, meaning, and action
04	The requested information has not been returned. Unless indicated differently in the following list of reason codes, all of the output areas for the information to be returned have been set to zeros. **Reason code** **Meaning** **00** The system could not determine the CHP type from the input CHPID. **01** The input CHPID is not configured. **02** The CHP type obtained from the input CHPID is not valid. **03** The input CHP type is invalid. **04** The input CHP_PARM is invalid. **05** The managed option (1) was specified for the CHP_PARM, but the CHP type is one that does not support dynamic channel path management. The default acronym and/or description is returned. **07** The input CHPID value is invalid. The value must be in the range `X'0000'` to `X'00FF'`.
06	The system was unable to obtain the requested information from the channel subsystem for this CHPID or the processor does not support returning the requested information.
08	Error in caller's parameters. **Reason code** **Meaning** **01** The caller specified an invalid ALET. **02** An error occurred in accessing the caller's parameter list. **03** A keyword that is not allowed to be specified with CHP_TYPE was specified; the keyword is only allowed when CHPID is specified. For instance, ATTR may only be specified when CHPID is specified.
0C	Recovery was entered.
20	Recovery was entered.

Chapter 101. IOSCMB — Locate the channel measurement block (CMB)

Description

The IOSCMB macro locates the channel measurement block (CMB) for a UCB and returns the data in either a 32 byte CMB format or a 64 byte ECMB format. This service eliminates the need for programs to know the format and location of the CMB.

Environment

Environmental factor	Requirement
Minimum authorization:	Supervisor state, zero PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts. If the caller is disabled, the parameter list (including any data areas pointed to from the parameter list) must be in fixed or DREF storage.
Locks:	The caller is not required to hold any locks on entry.
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

If the invoker is disabled, the parameter list, which includes any data areas pointed to by the parameter list, must reside in fixed or DREF storage.

Input register information

Before issuing the IOSCMB macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a 36–word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Performance implications

None.

Syntax

The IOSCMB macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCMB.
IOSCMB	
ƀ	One or more blanks must follow IOSCMB.
GET	**Default**: GET
,UCBPTR=*ucbptr_addr*	*ucbptr_addr*: Symbol, RX-type address, or register (2) - (12).
,CMBAREA=*cmbarea_addr*	*cmbarea_addr*: Symbol, RX-type address, or register (2) - (12).
,CMBLEN=64	**Default**: CMBLEN=64
,CMBLEN=32	
,ECMXAREA=NONE	**Default**: NONE
,ECMXAREA=*ecmxarea*	*ecmxarea*: Symbol, RS-type address, or register (2) - (12).
,ECMXLEN=*ecmxlen*	*ecmxlen*: Symbol, RS-type address, or register (2) - (12).

Syntax	Description
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).

Parameters

The parameters are explained below:

GET
Requests that the system locates the channel measurement block (CMB) for a UCB and return the data in either the old CMB format or the new ECMB format.

,UCBPTR=*ucbptr_addr*
Specifies a fullword containing the address of the UCB common segment whose CMB is to be returned.

,CMBAREA=*cmbarea_addr*
Specifies the address of a area to hold the measurement block being returned. The area can be either 32–bytes or 64–bytes, depending on what you specify for CMBLEN.

,CMBLEN=64
,CMBLEN=32
Specifies whether the area pointed to by CMBAREA is:

- 64 bytes and the channel measurement block info is to be returned in ECMB format, mapped by IRAECMB.
- 32 bytes and the channel measurement block information is to be returned in CMB format, mapped by IRACMB

,ECMXAREA=NONE
,ECMXAREA=*ecmxarea*
An optional input that specifies the address of an area of storage which is to receive the channel measurement extension information. This area can be mapped by the IOSDECMX macro, and the area must be at least the size specified in the IOCECMXLen field in the IECDIOCM mapping.

To code: Specify the RS-type address, or address in register (2) - (12), of a 4-byte field.

Default: NONE

,ECMXLEN=*ecmxlen*
When a value other than NONE is specified in ECMXAREA, a required input that specifies the length of the ECMXAREA. To receive all relevant data, the caller must specify an area at least the size specified in the IOCECMXLen field in the IECDIOCM mapping.

To code: Specify the RS-type address, or address in register (2) - (12), of a fullword.

,RETCODE
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

Return and reason codes

Table 95. Return and reason codes for the IOSCMB macro

Hexadecimal return code	Hexadecimal reason code	Meaning and action
00	- -	**Meaning**: Successful completion of the IOSCMB request. **Action** None.
08	01	**Meaning**: The IOSCMB request could not complete. No CMB was assigned for the device. **Action**: None; do not reissue this macro.

Chapter 102. IOSCMXA — Obtain address of the UCB common extension segment

Note: The UCBLOOK macro is the preferred programming interface.

Description

The IOSCMXA macro obtains the address of the UCB common extension segment. To map the UCB common extension segment, use the UCBCMEXT DSECT of the IEFUCBOB mapping macro.

Note: If you input a captured UCB address, you receive the address of the captured UCB common extension segment.

The IOSCMXA macro provides faster performance than the UCBLOOK macro; however, if the caller uses UCBLOOK to obtain several addresses in the same invocation, UCBLOOK might provide better performance than an IOSCMXA macro and an IOSUPFA macro. The UCBLOOK macro also validates input parameters and provides recovery. However, UCBLOOK cannot be used to obtain a captured UCB common extension address because UCBLOOK returns only actual UCB addresses.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	The input parameter must be in the primary address space. If the caller is disabled, the parameter list must reside in fixed or disabled reference (DREF) storage.

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be deleted. (If the caller issues a UCBLOOK macro with the PIN parameter to pin the UCB, use the UCBLOOK UCBCXPTR parameter rather than the IOSCMXA macro.)

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

None.

Input register information

Before issuing the IOSCMXA macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if the return code is 08

1
Used as a work register by the system

2-13
Unchanged

14
Return address

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

Performance implications

None.

Syntax

The standard form of the IOSCMXA macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCMXA.
IOSCMXA	
ƀ	One or more blanks must follow IOSCMXA.

Syntax	Description
UCBPTR=*ucbptr addr*	*ucbptr addr*: RX-type address or register (2) - (12).
,UCBCXPTR=*ucbcxptr addr*	*ucbcxptr addr*: RX-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

UCBPTR=*ucbptr addr*
Specifies the address of a fullword field that contains the address of the UCB common segment. This address must be for the UCB, and not for a copy of the UCB.

,UCBCXPTR=*ucbcxptr addr*
Specifies the address of a fullword field in which the system returns the address of the UCB common extension segment. Use the UCBCMEXT DSECT of the IEFUCBOB mapping macro to map the UCB common extension segment.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) — (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2) — (12).

ABEND codes

None.

Return and reason codes

When the IOSCMXA macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00		**Meaning**: Successful completion. **Action**: None.
08	03	**Meaning**: Program error. The UCB address provided by the caller parameter does not represent a valid UCB. **Action**: Correct the UCB address and reissue the macro.

IOSCMXA - List form

Use the list form of the IOSCMXA macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros.

Syntax

The list form of the IOSCMXA macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCMXA.
IOSCMXA	
ƀ	One or more blanks must follow IOSCMXA.
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default:** 0D

Parameters

The parameters are explained under the standard form of the IOSCMXA macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOSCMXA macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IOSCMXA - Execute form

Use the execute form of the IOSCMXA macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSCMXA macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCMXA.
IOSCMXA	
ƀ	One or more blanks must follow IOSCMXA.
UCBPTR=*ucbptr addr*	*ucbptr addr*: RX-type address or register (2) - (12).
,UCBCXPTR=*ucbcxptr addr*	*ucbcxptr addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained under the standard form of the IOSCMXA macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the IOSCMXA macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

Chapter 103. IOSCMXR — Obtain address of the UCB common extension segment

Description

Use the IOSCMXR macro to obtain the address of the unit control block (UCB) common extension segment. To map the UCB common extension segment, use the UCBCMEXT DSECT of the IEFUCBOB mapping macro.

Note: If you supply a captured UCB as input, you receive the address of the captured UCB common extension segment.

UCBLOOK and IOSCMXA macros also provide this function. However, IOSCMXR provides an alternative for passing parameters (that is, in general purpose register (GPR) 1 rather than in a parameter list). Also, UCBLOOK returns only actual, not captured, UCB addresses. For guidance about obtaining UCB information, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	None.

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

If you input a captured UCB address, the UCB must be captured in the primary address space.

Input register information

Before issuing the IOSCMXR macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of UCB common segment of the UCB

Before issuing the IOSCMXR macro, the caller does not have to place any information into any access register (AR).

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the UCB common extension

2-13
Unchanged

14-15
Used as work registers by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IOSCMXR macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCMXR.
IOSCMXR	
ƀ	One or more blanks must follow IOSCMXR.
MF=(S)	**Default**: S

Parameters

The parameters are explained as follows:

MF=(S)
Specifies the standard form of the macro. This parameter is optional.

ABEND codes

None.

Return and reason codes

None.

Chapter 104. IOSCUINF — Control unit information service

Description

The IOSCUINF macro provides data of the specific control unit according to requests and also gives user the ability to reset high watermark measurements.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or Supervisor state. Any PSW key.
Dispatchable unit mode:	Task or SRB mode
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts

Programming requirements

None.

Restrictions

- Callers cannot hold any locks that prevent the IOSCUINF service from obtaining the IOSYNCH lock.
- The LINKAGE=BRANCH option is limited to callers that meet all of the following criteria:
  - supervisor state key 0
  - 31-bit addressing mode
  - Primary ASC mode
  - The parameter list resides in fixed or DREF storage
- No information is returned either by a control unit number specified for the CU keyword or by a token NED specified for the TOKENNED keyword in the CTC device.

Input register information

Before issuing the IOSCUINF macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

Performance implications

None.

Syntax

The standard form of the IOSCUINF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCUINF.
IOSCUINF	
ƀ	One or more blanks must follow IOSCUINF.
CU=*cu*	*cu*: The name (RS-type) of a halfword input.
TOKENNED=*tokenned*	*tokenned*: RX-type address or register (2) - (12).
CLASS	
,CUCLASS=ALL|TAPE|COMM|DASD|DISP|UREC|CHAR	**Default**: ALL
GROUP	
,CUGROUP=PAV|HYPERPAV	**Default**: PAV **Note:** Specify only one of the above keywords: CU, TOKENNED, CLASS, or GROUP.
,OUTPUT_AREA=*output_area*	*output_area*: RS-type address or register (2) - (12).

Syntax	Description
,RESET_MEASURES	**Default**: None
,PATHINFO	**Default**: None
,LINKAGE=SYSTEM|BRANCH	**Default**: SYSTEM
,OUTPUT_VERSION=2	**Default**: None
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=*plistver*|IMPLIED_VERSION	**Default**: IMPLIED_VERSION
,MF=S	**Default**: MF=S
,MF=(L,*mfctrl*,*mfattr*, 0D)	
,MF=(E,*mfctrl*,COMPLETE)	

Parameters

In the following set of mutually exclusive keywords, only one keyword must be specified.

CU=*cu*
The name (RS-type) of a halfword input that contains the number of the physical control unit that the data is to be retrieved from.

TOKENNED=*tokenned*
The name (RS-type) of a 32-character input of the token NED that is the worldwide-unique identifier for the subsystem to which information is to be returned.

CLASS
Indicates that a control unit class is specified.

,CUCLASS=ALL|TAPE|COMM|DASD|DISP|UREC|CHAR
An optional keyword input that specifies a control unit class for which the data is to be retrieved.

CUCLASS=ALL
Requests data for all control units in the I/O configuration except for those in the CTC device class.

CUCLASS=TAPE
Requests data for TAPE device class.

CUCLASS=COMM
Requests data for communications device class.

CUCLASS=DASD
Requests data for DASD device class.

CUCLASS=DISP
Requests data for display device class.

CUCLASS=UREC
Requests data for unit record device class.

CUCLASS=CHAR
Requests data for character reader device class.

Default: ALL

GROUP
Indicates that a group is specified.

,CUGROUP=PAV|HYPERPAV
An optional keyword input that specifies a control unit group for which the data is to be retrieved.

CUGROUP=PAV
Requests data for parallel access volume (PAV) control units.

CUGROUP=HYPERPAV
Requests data for hyper parallel access volume (HYPERPAV) control units.

Default: PAV.

End of the mutually exclusive keywords.

,OUTPUT_AREA=*output_area*
A required pointer output that contains the address of the output area returned by the IOSCUINF service. The area is mapped by the IOSDCUIN data area. Storage for the OUTPUT_AREA is obtained by the service and must be released by the caller.

Callers must use the appropriate length fields defined in the IOSDCUIN mapping macro to ensure accuracy in navigating through out the output area. Avoid using the length of the DSECT (using L '*xxxxxxxx* of any structure). IBM strongly suggests that references to the following list of equate symbols be replaced by the appropriate fields:

Equate symbol	Field to use
CUIN_LEN	CUIN_TOTAL_LENGTH
CUINENTRY_LEN	CUIN_ENTRY_LENGTH
CUIN_PATHINFO_HEADER_LEN	CUIN_PI_HdrLen
CUIN_PATHINFO_LEN	CUIN_PI_EntLen

Comments in the IOSDCUIN mapping macro describe how to base each DSECT and how to advance the structure when multiple entries are returned.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) (ASM only).

,RESET_MEASURES
Indicates that the high watermarks are to be reset for those control units that the data was collected for.

Default: NONE.

,PATHINFO
Indicates that path information is to be returned for the control units for which data was collected. The path information includes the CU number, the interface id, the tag, the CHPID, the link address, and the WWPN for each path attached to the control unit.

Default: NONE.

,LINKAGE=SYSTEM|BRANCH
An optional keyword input that indicates whether a program call is issued or a branch-entry linkage is generated for the routine invocation.

Default: SYSTEM.

LINKAGE=SYSTEM
Requests program call invocation.

LINKAGE=BRANCH
Requests branch-entry invocation. See "Restrictions" on page 1043 for the restrictions on branch-entry invocation.

,OUTPUT_VERSION=2
An optional input parameter that specifies the version of the output that the caller desires. OUTPUT_VERSION=2 indicates that the service is to return additional statistics in the OUTPUT_AREA about control units running with HYPERPAV=XPAV (also known as SuperPAV mode).

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) (ASM only).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or address in register (2) - (12) (ASM only).

,PLISTVER=*plistver*|IMPLIED_VERSION
An optional byte input decimal value in the "1-1" range that specifies the macro version. PLISTVER is the only parameter allowed on the list form of MF. This parameter determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values can be:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- 1

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE
An optional keyword input that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the servIice. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning for each return and reason code.

Table 96. Return and Reason Codes for the IOSCUINF Macro

Return Code	Reason Code	Meaning and Action
00	None	**Explanation**: IOSCUINF request successful.
04	None	**Explanation**: Find no control units that match the requested criteria.
08	xxxx0001	**Explanation**: Can not use this service in AR ASC mode. **Note:** The OUTPUT_AREA was not returned by the service and should not be released by the caller.
08	xxxx0002	**Explanation**: The selection code is not valid. **Note:** The OUTPUT_AREA was not returned by the service and should not be released by the caller.
20	None	**Explanation**: An unexpected error occurred. **Note:** The OUTPUT_AREA was not returned by the service and should not be released by the caller.

Chapter 105. IOSCUMOD — IOS control unit entry build service

Description

IOSCUMOD is a prototype module, to be used by manufacturers for creating an IOSTnnn load module and for building the control unit model table.

Programming requirements

On the first invocation of the IOSCUMOD macro, it includes the parameters listed below in the manufacturer's module.

Restrictions

None.

Performance implications

None.

Syntax

The IOSCUMOD macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCUMOD.
IOSCUMOD	
ƀ	One or more blanks must follow IOSCUMOD.
MANF=*chpid*	*manf*: Symbol up to 3 characters long.
,DEVT=*devt*	*devt*: Symbol up to 6 characters long.
,MODN=*devt*	*modn*: Symbol up to 3 characters long.
,MASK1=*mask1*	*mask1*: 2-byte hex symbol.

Syntax	Description
,MASK2=*mask2*	*mask2*: 2-byte hex symbol.
,MASK3=*mask3*	*mask3*: 2-byte hex symbol.
,MASK4=*mask4*	*mask4*: 2-byte hex symbol.
,DCM_SUPPORTED=YES	**Default**: YES
,DCM_SUPPORTED=NO	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSCUMOD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

MANF=*manf*
Manufacturer ID that was provided with the node descriptor.

,DEVT=*devt*
Device type ID that was provided with the node descriptor. If a 4-character device type is entered, the two leading fields will be set to blanks.

,MODN=*modn*
Model number ID that was provided with the node descriptor. If NULL, then the model field will be set to all blanks. Othwerwise, leading zeroes must be coded.

,MASK1=*mask1*
,MASK2=*mask2*
,MASK3=*mask3*
,MASK4=*mask4*
Hex equivalent of the masks defined. 4 hex digits must be provided.

The tag field of the node descriptor uniquely identifies the power/service boundaries of most control units. Although this is true in most cases, it is not architected that way, and different control units represent this information in different ways.

In order to be able to interpret a control units tag, each control unit will provide four 2-byte masks.

Each 2 byte mask will be ANDed against the tag field of the control unit's Node Descriptor to extract a unique indicator of the different service boundary in the control unit. The first (high order) mask will indicate the most significant single point of failure to avoid (For example, Cluster), the second mask will indicate the most significant single failure to avoid (e.g. I/O bay), and so on until the fourth mask.

There is no requirement for the masks to represent specific components of the control (e.g. Cluster vs. I/O Bay vs. Port card). The only requirement is that the masks are ordered from the most significant point of failure to least. If not all four masks are significant, they should be set to binary zeros and must be the last mask(s) of the four.

,DCM_SUPPORTED=YES
,DCM_SUPPORTED=NO
Indicates that the control unit does or does not support dynamic channel path management. Control units which support ESCON interfaces and are completely non-synchronous should be capable of being supported by DCM. Control units which transfer data synchronously from the media, or remain connected to the channel while waiting for data to transfer between the media and the cache (or channel), are not supported. The default is YES.

ABEND codes

None.

Return and reason codes

None.

System macros require High Level Assembler. For more information about assembler language programming, see High Level Assembler and Toolkit Feature in IBM Documentation (www.ibm.com/docs/en/hla-and-tf/1.6).

Using this information also requires you to be familiar with the operating system and the services that programs running under it can invoke.

Chapter 106. IOSDCXR — Obtain address of the device class extension segment

Description

Use the IOSDCXR macro to obtain the address of the unit control block (UCB) device class extension segment. For example, the DASD device class extension segment is mapped by the IECDDCE macro and the tape device class extension segment is mapped by the IECUCBCX macro.

Note: If you supply a captured UCB as input, you receive the address of the captured UCB device class extension segment except under either of the following conditions:

- The unit information module (UIM) indicates that the device class extension segment can reside above 16 megabytes independent of the rest of the UCB.
- The UIM indicates that a single device class extension segment can be shared by multiple UCBs.

In these cases, you receive the address of the actual, not captured, UCB device class extension segment.

Other macros provide addresses to other UCB segments. For example, UCBLOOK, IOSCMXA, and IOSCMXR provide the address of the UCB common extension segment. For guidance about obtaining UCB information, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	None.

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

If you input a captured UCB address, the UCB must be captured in the primary address space.

Input register information

Before issuing the IOSDCXR macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of UCB common segment

Before issuing the IOSDCXR macro, the caller does not have to place any information into any access register (AR).

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the UCB device class extension segment if the UCB has a device class extension segment. Zero if the UCB does not have a device class extension segment.

2-13
Unchanged

14-15
Used as work registers by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IOSDCXR macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSDCXR.
IOSDCXR	
ƀ	One or more blanks must follow IOSDCXR.

Syntax	Description
MF=(S)	**Default**: S

Parameters

The parameters are explained as follows:

MF=(S)
Specifies the standard form of the macro. This parameter is optional.

ABEND codes

None.

Return and reason codes

None.

Chapter 107. IOSENQ — IOS ENQ service

Description

IOSENQ allows you to perform ENQs and DEQs on certain I/O Supervisor (IOS) resources. Currently, the following functions can be serialized:

- Dynamic channel path management
- Dynamic I/O processing

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state. Zero PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks may be held.
Control parameters:	Must be in the primary address space.

Programming requirements

The caller should include the IOSDENQ macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

The caller must not have a pending ENQ for the same resource managed by this service.

Input register information

Before issuing the IOSENQ macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Unpredictable (Used as a work register by the system)

2-13
Unchanged

14
Unpredictable (Used as a work register by the system)

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Unpredictable (Used as work registers by the system)

2-13
Unchanged

14-15
Unpredictable (Used as work registers by the system)

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IOSENQ macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSENQ.
IOSENQ	
ƀ	One or more blanks must follow IOSENQ.
RESOURCE=DYNCHPID	
,REQUEST=ENQ	
,STATE=SHARED	
,STATE=EXCLUSIVE	
,COND=NO	
,COND=YES	
,WAITTIME=*waittime*	Required choice with REQUEST=ENQ,COND=YES.

Syntax	Description
,WAITTIME=SYSTEM_DEFINED	Default: SYSTEM_DEFINED.
,REQUEST=DEQ	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	*list addr*: RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSENQ macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

RESOURCE=DYNCHPID
RESOURCE=NOT_SUPPORTED
A required input parameter. Indicates that this request will deal with the dynamic channel path management ENQ resource.

,REQUEST=ENQ
,REQUEST=DEQ
A required input parameter.

,REQUEST=ENQ
Indicates that the request is to perform an ENQ operation.

STATE=SHARED
STATE=EXCLUSIVE
A required input parameter if REQUEST=ENQ is specified.

STATE=SHARED
Indicates that the ENQ should be obtained in shared state.

STATE=EXCLUSIVE
Indicates that the ENQ should be obtained in exclusive state.

COND=NO
COND=YES
A required input parameter if REQUEST=ENQ is specified.

COND=NO
Indicates that this is not a conditional ENQ. Control will only be returned to the caller when the ENQ is held.

COND=YES
Indicates that this is a conditional ENQ. If the ENQ cannot be obtained within the given length of time, processing is ended, and a return code indicating this situation is provided to the caller.

,WAITTIME=*waittime*
,WAITTIME=SYSTEM_DEFINED
A required input parameter if REQUEST=ENQ,COND=YES is specified.

,WAITTIME=*waittime*
The name (RS-type), or address in register (2)-(12), of a fullword input that specifies the maximum time in hundredths of seconds that the system is to wait for the ENQ to be obtained. A value of 0, or omitting this parameter, results in the system using a pre-determined wait time. The value is treated as a 32-bit unsigned number.

,WAITTIME=SYSTEM_DEFINED
The pre-determined default time in hundredths of seconds that the system is to wait for the ENQ to be obtained.

,REQUEST=DEQ
Indicates that the request is to perform a DEQ operation. A warning return code will result if the ENQ is not held.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX

- A decimal value of 1

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

The caller may get the following abend codes:

Table 97. ABEND Codes for the IOSENQ Macro

ABEND Code	Meaning and Action
0C4-4	**Meaning:** Your program was not in supervisor state with PSW key 0. **Action:** Call IOSENQ only when in supervisor state with PSW key 0.
B78-8	**Meaning:** Your program was in problem state with PSW key 8-15. **Action:** Call IOSENQ only when in supervisor state with PSW key 0.

Return and reason codes

Macro IOSDENQ provides equate symbols for the return and reason codes.

When the IOSENQ macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

Table 98. Return and Reason Codes for the IOSENQ Macro

Hexadecimal Return Code	Reason Codes, Meaning and Action
00	**Equate Symbol:** IOSENQRc_OK **ENQ** The ENQ is held. **DEQ** The DEQ is held.
04	**Equate Symbol:** IOSENQRc_Warn **Reason Code** **Meaning/Action** **01** **Equate Symbol:** IOSENQRsnEnqAlreadyHeld **Meaning:** For ENQ, this task already holds the ENQ. It could be held either in shared or exclusive state. **Action:** Avoid using the IOSENQ REQUEST=ENQ function when you already hold the requested ENQ. **02** **Equate Symbol:** IOSENQRsnEnqNotHeld **Meaning:** DEQ was requested but the caller did not have the ENQ. **Action:** None required.
08	**Equate Symbol:** IOSENQRc_InvParm **Reason Code** **Meaning/Action** **01** **Equate Symbol:** IOSENQRsnBadRequest **Meaning:** An incorrect request was specified. **Action:** Check for storage overlays. **02** **Equate Symbol:** IOSENQRsnBadResource **Meaning:** An incorrect resource was requested. **Action:** Check for storage overlays.
0C	**Equate Symbol:** IOSENQRc_Env **Reason Code** **Meaning/Action** **01** **Equate Symbol:** IOSENQRsnCouldNotGetENQ **Meaning:** On a conditional ENQ request, the ENQ could not be obtained within the specified time period. **Action:** Specify a longer time period, or request the ENQ unconditionally.

Chapter 108. IOSFBA — IOS fixed block architecture service

Description

IOSFBA is used to manage (allocate and unallocate) and perform I/O (read and write) to allocated fixed block architecture (FBA) devices. IOSFBA has the following functions:

ALLOCATE
Allocates one or more FBA devices to be used with the READ or WRITE IOSFBA service.

QUERY
Provides information about the requested devices. The caller provides a list of device numbers for which the service returns device attributes.

READ
Initiates read operations for one or more devices as described in the supplied device I/O list.

WRITE
Initiates write operations for one or more devices as described in the supplied device I/O list.

ERASE
Initiates erase operations for a contiguous area for one or more devices as described in the supplied device I/O list.

CLEANUP
Cleans up resources associated with an I/O token.

UNALLOCATE
Unallocates one or more FBA devices that were previously allocated with this service.

Note: Mapping macros for the IOS fixed block architecture services are contained in IOSDFBA.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, PSW Key 0.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks may be held.
Control parameters:	

Programming requirements

None.

Restrictions

The invoker must have SAF authorization to facility 'IOSFBA'. Specifically, the invoker must have UPDATE authority to facility class 'IOSFBA'.

Input register information

Before issuing the IOSFBA macro, the caller must ensure that general register 13 contains the address of a 72 byte save area (for AMODE(31) callers) or 216 byte save area (for AMODE(64) callers). The save area must be in primary storage in the first 2GB of storage. The caller does not have to place any information into any other general purpose register (GPR) unless using it in register notation for a particular parameter or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Unpredictable (Used as a work register by the system)

2-13
Unchanged

14
Unpredictable (Used as a work register by the system)

15
Return code

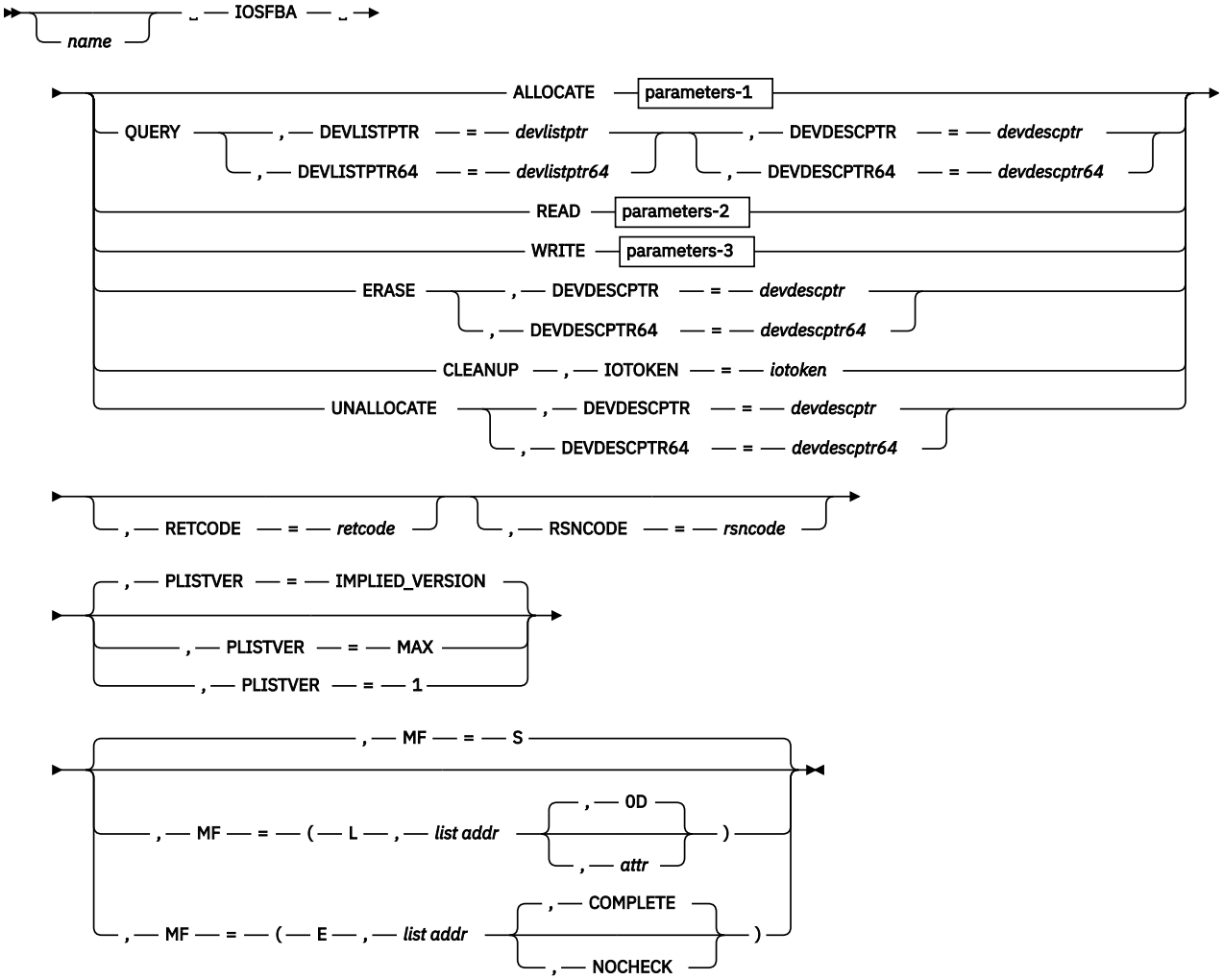
Performance implications

None.

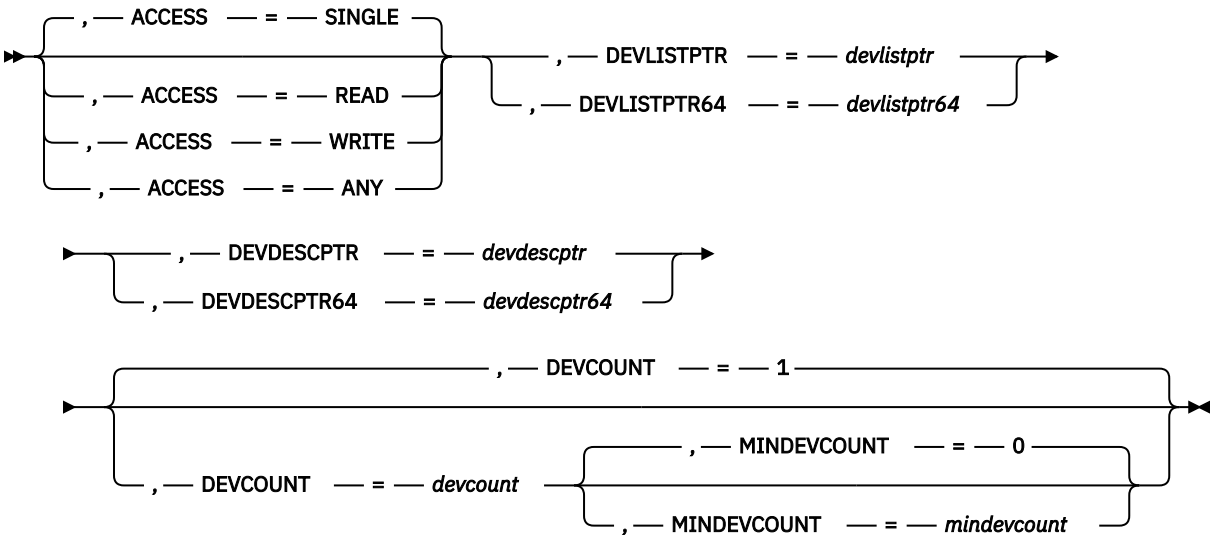
Syntax

The IOSFBA macro is written as follows:

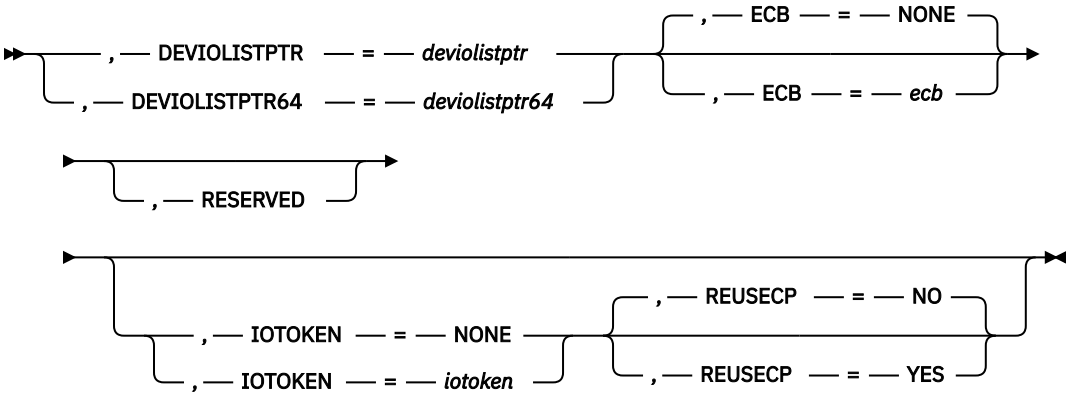
main diagram



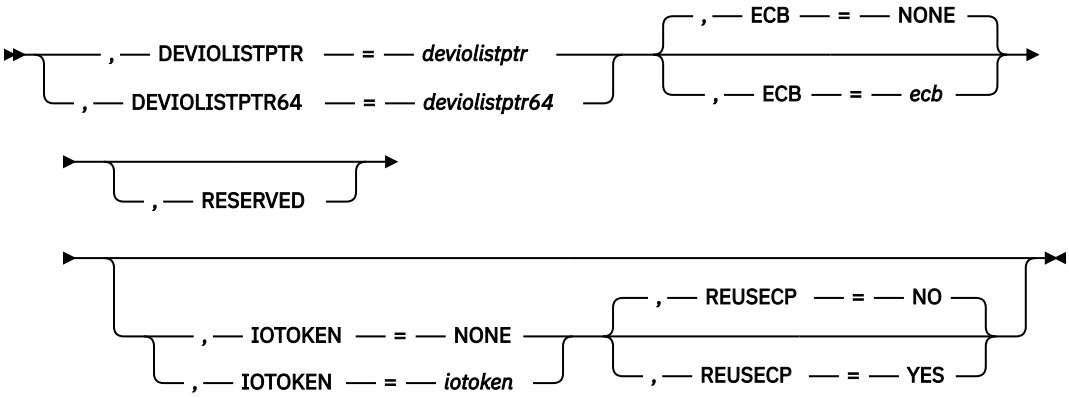
parameters-1



parameters-2



parameters-3



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSFBA macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,ACCESS=SINGLE
,ACCESS=READ
,ACCESS=WRITE
,ACCESS=ANY
When ALLOCATE is specified, ACCESS is an optional parameter that indicates the type of allocation that should be performed. The default is ACCESS=SINGLE.

,ACCESS=SINGLE
The FBA device is allocated for use by a single z/OS system. Note that one or more distributed systems can access this device in addition to the single z/OS system. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

,ACCESS=READ
The FBA device is allocated for READ only on the system where this request is received. This device can also be allocated on another z/OS system by requesting ACCESS=WRITE. Note that one or more distributed systems can access this device in addition to the pair of z/OS systems that performed the z/OS allocations. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

,ACCESS=WRITE
The FBA device is allocated for WRITE only on the system where this request is received. This device can also be allocated on another z/OS system by requesting ACCESS=READ. Note that one or more distributed systems can access this device in addition to the pair of z/OS systems that performed the z/OS allocations. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

,ACCESS=ANY
The FBA device is allocated once on each z/OS system. Note that one or more distributed systems can access this device in addition to the z/OS systems that performed the z/OS allocations. Access from distributed systems is controlled through the use of LUN masking or using the persistent reserve from the distributed side.

ALLOCATE
A required input parameter that allocates the devices as specified in the device list (FBADL DSECT in the IOSDFBA macro). This request, if successful, allocates the requested number of devices within a sysplex for the caller's use.

To code: Specify a value.

,CLEANUP
A required input parameter that cleans up resources for the input IOTOKEN. During IOSFBA READ or IOSFBA WRITE invocations, one or more IOTOKEN areas may have been used to allow for efficiency. IOSFBA CLEANUP must be invoked once for each IOTOKEN that was used during processing.

To code: Specify a value.

,DEVCOUNT=*devcount*
,DEVCOUNT=1
When ALLOCATE is specified, DEVCOUNT is an optional input parameter that contains the number of devices that should be allocated from the devices specified in the device list (FBADL) addressed by the DEVLISTPTR or DEVLISTPTR64 parameter. DEVCOUNT indicates the maximum number of devices to allocate. DEVCOUNT should be less than or equal to the number of devices specified in the device list. The default is 1.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,DEVDESCPTR=*devdescptr*
When ALLOCATE is specified, DEVDESCPTR is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).

The device descriptor area contains the addresses of a device descriptor entry for each device successfully allocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro). The device descriptor entry is required input for the IOSFBA READ and IOSFBA WRITE services.

When using the IOSFBA READ or IOSFBA WRITE services, the device descriptor entry is required input for each device that is read from or written to. Refer to the DEVIOLISTPTR or DEVIOLISTPTR64 parameter for the IOSFBA READ and/or IOSFBA WRITE service for more information.

The device descriptor address is an input parameter for the IOSFBA UNALLOCATE service. The UNALLOCATE service unallocates the devices contained in the device descriptor area.

The caller is responsible for freeing or releasing this storage after the devices have been UNALLOCATED. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVDESCPTR=*devdescptr*
When QUERY is specified, DEVDESCPTR is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).

The device descriptor area contains the addresses of a device descriptor entry for each device successfully queried. The device descriptor entry contains specific device information discovered by the QUERY service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The caller is responsible for freeing or releasing this storage after the devices have been queried. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVDESCPTR=*devdescptr*
When ERASE is specified, DEVDESCPTR is a required input parameter of the input device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro). The device descriptor area is obtained by the IOSFBA ALLOCATE service.

The device descriptor area contains the addresses of the device descriptor entry for each device to be unallocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The status of the erase request for each device is obtained by checking the FBADDL_EraseFailed and FBADDL_NoEraseAttempted indicators in the FBADDL for each device. If FBADDL_EraseFailed is indicated, the FBADDE_COD and FBADDE_RCOD fields contain information about the I/O failure.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVDESCPTR=*devdescptr*
When UNALLOCATE is specified, DEVDESCPTR is a required input parameter of the input device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro). The device descriptor area is obtained by the IOSFBA ALLOCATE service.

The UNALLOCATE service unallocates the devices contained in the device descriptor area, specifically each of the device descriptor entries.

The device descriptor area contains the addresses of the device descriptor entry for each device to be unallocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The status of the unallocation request for each device is obtained by checking the return code and reason code contained in the device descriptor entry (FBADDE). The return code and reason code contain the dynamic allocation (SVC 99) return and reason code or an IOSFBA service return and reason code.

The caller is responsible for freeing or releasing this storage after the devices have been UNALLOCATED. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVDESCPTR64=*devdescptr64*

When ALLOCATE is specified, DEVDESCPTR64 is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).

The device descriptor area contains the addresses of a device descriptor entry for each device successfully allocated. The device descriptor entry contains specific device information discovered during by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro). The device descriptor entry is required input for the IOSFBA READ and IOSFBA WRITE services.

When using the IOSFBA READ or IOSFBA WRITE services, the device descriptor entry is required input for each device that is read from or written to. Refer to the DEVIOLIST parameter for the IOSFBA READ and/or IOSFBA WRITE service for more information.

The device descriptor address is an input parameter for the IOSFBA UNALLOCATE service. The UNALLOCATE service unallocates the devices contained in the device descriptor area.

The caller is responsible for freeing or releasing this storage after the devices have been UNALLOCATED. The storage area must be freed using the IARST64 service.

Note: The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVDESCPTR64=*devdescptr64*

When QUERY is specified, DEVDESCPTR64 is a required output parameter of the output device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro).

The device descriptor area contains the addresses of a device descriptor entry for each device successfully queried. The device descriptor entry contains specific device information discovered by the QUERY service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The caller is responsible for freeing or releasing this storage after the devices have been queried. The subpool and length of the storage are contained in the device descriptor area. The device descriptor area must be freed using either STORAGE RELEASE or FREEMAIN macro invocation.

Note: The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVDESCPTR64=*devdescptr64*

When ERASE is specified, DEVDESCPTR64 is a required input parameter of the input device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro). The device descriptor area is obtained by the IOSFBA ALLOCATE service.

The UNALLOCATE service unallocates the devices contained in the device descriptor area, specifically each of the device descriptor entries.

The device descriptor area contains the addresses of the device descriptor entry for each device to be unallocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The status of the erase request for each device is obtained by checking the FBADDL_EraseFailed and FBADDL_NoEraseAttempted indicators in the FBADDL for each device. If FBADDL_EraseFailed is indicated, the FBADDE_COD and FBADDE_RCOD fields contain information about the I/O failure.

Note: The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVDESCPTR64=*devdescptr64*
When UNALLOCATE is specified, DEVDESCPTR64 is a required input parameter of the input device descriptor area mapped by the FBADDL (defined in the IOSDFBA macro). The device descriptor area is obtained by the IOSFBA ALLOCATE service.

The UNALLOCATE service unallocates the devices contained in the device descriptor area - specifically each of the device descriptor entry.

The device descriptor area contains the addresses of the device descriptor entry for each device to be unallocated. The device descriptor entry contains specific device information discovered by the ALLOCATE service. The device descriptor entry is mapped by the FBADDE (defined in the IOSDFBA macro).

The status of the unallocation request for each device is obtained by checking the return code and reason code contained in the device descriptor entry (FBADDE). The return code and reason code contain the dynamic allocation (SVC 99) return and reason code or an IOSFBA service return and reason code.

The caller is responsible for freeing or releasing this storage after the devices have been UNALLOCATED. The device descriptor area must be freed using the IARST64 service.

Note: The DEVDESCPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVIOLISTPTR=*deviolistptr*
When READ is specified, DEVIOLISTPTR is a required input parameter of the input device I/O list mapped by the FBADIOL (defined in the IOSDFBA macro). The device I/O list specifies the number of devices that will participate in the IOSFBA READ service. Additionally, the device I/O list contains a pointer to the device I/O entry for each device (mapped by the FBADIOE, defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that was returned as part of the device descriptor area by the IOSFBA ALLOCATE service), a count of extent entries, the addresses of each extent entry (mapped by the FBAEE, defined in the IOSDFBA macro), and if required by the caller, the address of a status block or area (mapped by the FBAST, defined in the IOSDFBA macro). For a complete description, see the IOSDFBA macro.

The extent entry defines the parameters of the READ I/O operation for a given device. It defines the starting block number on the device, the number of blocks to transfer, and the storage address or addresses to place the information read from the FBA device.

The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA READ service.

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

Note: The DEVIOLISTPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVIOLISTPTR=*deviolistptr*
When WRITE is specified, DEVIOLISTPTR is a required input parameter of the input device I/O list mapped by the FBADIOL (defined in the IOSDFBA macro). The device I/O list specifies the number of devices that will participate in the IOSFBA READ service. Additionally, the device I/O list contains a device I/O entry for each device (mapped by the FBADIOE, defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that was returned as part of the device descriptor area by the IOSFBA ALLOCATE service), a count of extent entries, the addresses of each extent entry (mapped by the FBAEE, defined in the IOSDFBA macro), and if required by the caller, the address of a status block or area (mapped by the FBAST, defined in the IOSDFBA macro). For a complete description, see the IOSDFBA macro.

The extent entry defines the parameters of the WRITE I/O operation for a given device. It defines the starting block number on the device, the number of blocks to transfer, and the storage address or addresses to place the information read from the FBA device.

The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA WRITE service.

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

Note: The DEVIOLISTPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVIOLISTPTR64=*deviolistptr64*

When READ is specified, DEVIOLISTPTR64 is a required input parameter of the input device I/O list mapped by the FBADIOL (defined in the IOSDFBA macro). The device I/O list specifies the number of devices that will participate in the IOSFBA READ service. Additionally, the device I/O list contains a device I/O entry for each device (mapped by the FBADIOE, defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that was returned as part of the device descriptor area by the IOSFBA ALLOCATE service), a count of extent entries, the addresses of each extent entry (mapped by the FBAEE, defined in the IOSDFBA macro), and if required by the caller, the address of a status block or area (mapped by the FBAST, defined in the IOSDFBA macro). For a complete description, see the IOSDFBA macro.

The extent entry defines the parameters of the READ I/O operation for a given device. It defines the starting block number on the device, the number of blocks to transfer, and the storage address or addresses to place the information read from the FBA device.

The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA READ service.

Note: The DEVIOLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVIOLISTPTR64=*deviolistptr64*

When WRITE is specified, DEVIOLISTPTR64 is a required input parameter of the input device I/O list mapped by the FBADIOL (defined in the IOSDFBA macro). The device I/O list specifies the number of devices that will participate in the IOSFBA READ service. Additionally, the device I/O list contains a device I/O entry for each device (mapped by the FBADIOE, defined in the IOSDFBA macro).

The device I/O entry includes the address of the device descriptor entry (that was returned as part of the device descriptor area by the IOSFBA ALLOCATE service), a count of extent entries, the addresses of each extent entry (mapped by the FBAEE, defined in the IOSDFBA macro), and if required by the caller, the address of a status block or area (mapped by the FBAST, defined in the IOSDFBA macro). For a complete description, see the IOSDFBA macro.

The extent entry defines the parameters of the WRITE I/O operation for a given device. It defines the starting block number on the device, the number of blocks to transfer, and the storage address or addresses to place the information read from the FBA device.

The status block provides the caller with status information for the I/O to each device. A status block should be obtained and initialized to zeroes by the caller for each device that will participate in the IOSFBA WRITE service.

Note: The DEVIOLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVLISTPTR=*devlistptr*
When ALLOCATE is specified, DEVLISTPTR is a required input parameter of the input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

Note: The DEVLISPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVLISTPTR=*devlistptr*
When QUERY is specified, DEVLISTPTR is a required input parameter of the input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

For AMODE(31) callers, the storage area must be addressable in AMODE(31).

Note: The DEVLISPTR parameter is allowed only when not in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,DEVLISTPTR64=*devlistptr64*
When ALLOCATE is specified, DEVLISTPTR64 is a required input parameter of the input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

Note: The DEVLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,DEVLISTPTR64=*devlistptr64*
When QUERY is specified, DEVLISTPTR64 is a required input parameter of the input device list mapped by the FBADL (defined in the IOSDFBA macro). The FBADL specifies the number of devices, the device numbers, and others to allocate. (Refer to the FBADL for specific information.)

Note: The DEVLISTPTR64 parameter is allowed only when in AMODE 64 as indicated by the SYSSTATE macro.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte pointer field.

,ECB=*ecb*
,ECB=NONE
When READ is specified, ECB is an optional input parameter that contains the address that points to an optional ECB that is posted when all read operations are complete. If an ECB is not specified, control returns to the invoker when all read operations have completed. The default is NONE.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,ECB=*ecb*
,ECB=NONE
When WRITE is specified, ECB is an optional input parameter that contains the address that points to an optional ECB that is posted when all write operations are complete. If an ECB is not specified, control returns to the invoker when all write operations have completed. The default is NONE.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,ERASE
A required input parameter that erases a contiguous area of the device or devices as specified in the device list mapped by the FBADL DSECT (defined in the IOSDFBA macro). Erase writes null characters (X'00').

To code: Specify a value.

,IOTOKEN=*iotoken*
,IOTOKEN=NONE
When READ is specified, IOTOKEN is an optional input parameter that contains the address that points to an optional 32-byte area used by the IOSFBA service to store addresses and lengths of storage areas that can be reused in order to avoid system overhead of obtaining these resources on each call.

The invokers of IOSFBA can use as many unique 32-byte IOTOKEN areas as desired. When IOTOKEN is used,

- The area specified by the IOTOKEN should be initially cleared by the calling program before the first usage of the IOTOKEN area.
- Each IOTOKEN is used to represent storage areas for the life of an I/O request. The caller should not reuse an IOTOKEN until the I/Os initiated for it have completed. For synchronous callers, the IOTOKEN can be immediately reused. For asynchronous callers (when an ECB is provided), the caller must not reuse an IOTOKEN until the ECB has been posted.
- IOSFBA CLEANUP must be invoked for each IOTOKEN that was used to ensure that task related storage is released.

The default is NONE.

To code: Specify the RS-type address of a pointer field.

,IOTOKEN=*iotoken*
,IOTOKEN=NONE
When WRITE is specified, IOTOKEN is an optional input parameter that contains the address that points to an optional 32-byte area that is used by the IOSFBA service to store addresses and lengths of storage areas that can be reused in order to avoid system overhead of obtaining these resources on each call.

The invokers of IOSFBA can use as many unique 32-byte IOTOKEN areas as desired. When IOTOKEN is used,

- The area specified by the IOTOKEN should be initially cleared by the calling program before the first usage of the IOTOKEN area.
- Each IOTOKEN is used to represent storage areas for the life of an I/O request. The caller should not reuse an IOTOKEN until the I/Os initiated for it have completed. For synchronous callers, the IOTOKEN can be immediately reused. For asynchronous callers (when an ECB is provided), the caller must not reuse an IOTOKEN until the ECB has been posted.
- IOSFBA CLEANUP must be invoked for each IOTOKEN that was used to ensure that task related storage is released.

The default is NONE.

To code: Specify the RS-type address of a pointer field.

,IOTOKEN=*iotoken*
When CLEANUP is specified, IOTOKEN is a required input parameter that contains the address that points to a 32-byte area that is used by the IOSFBA service to store addresses and lengths of storage areas that can be reused in order to avoid system overhead of obtaining these resources on each call.

To code: Specify the RS-type address of a pointer field.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this is an RS-type address or an address in register (1)-(12).

,attr
An optional 1 to 60 character input string used to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

,MINDEVCOUNT=*mindevcount*
,MINDEVCOUNT=0
When ALLOCATE is specified, MINDEVCOUNT is an optional input parameter that indicates the minimum number of devices that must be allocated to fulfill this allocate request. The devices are specified in the device list (FBADL) addressed by the DEVLISTPTR or DEVLISTPTR64 parameter. This number should be less than or equal to the number specified in DEVCOUNT.

If the caller requests DEVCOUNT=*x* and MINDEVCOUNT=*y*, the IOSFBA service attempts to allocate the requested number of devices (as specified by the DEVCOUNT=*x* parameter). If '*x*' devices are not available to be allocated, IOSFBA ALLOCATE service attempts to allocate as many devices that are available. The ALLOCATE request is considered successful if at least '*y*' devices are allocated (as specified by the MINDEVCOUNT=*y* parameter). The ALLOCATE request is considered unsuccessful if '*y*' devices (as specified by the MINDEVCOUNT=y parameter) are not allocated and a return code is set indicating the ALLOCATE request failed since the minimum number of devices could not be allocated.

The count or number of devices that have been allocated is contained in the device descriptor area (mapped by the FBADDL).

If this keyword is omitted or specified as 0, the MINDEVCOUNT is assumed to be the value specified on the DEVCOUNT keyword. The default is 0.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,QUERY
A required input parameter, queries information about a list of devices. Information is returned for each device in the list of devices identified in the DEVLIST keyword.

To code: Specify a value.

,READ
A required input parameter, reads information from one or more FBA devices as specified by the device I/O list (DEVIOLIST parameter).

To code: Specify a value.

,RESERVED
When READ is specified, RESERVED is an optional input parameter indicating that the device or devices for this READ operation may be serialized by persistent reserve from a distributed client. When this keyword is specified, z/OS I/O operations are permitted to the devices while the persistent reserve is held. This keyword should only be used when the invoking program is coordinating I/O activity between z/OS and the distributed client that owns the persistent reserve. The default is NONE.

To code: Specify a value.

,RESERVED
When WRITE is specified, RESERVED is an optional input parameter that indicates that the device or devices for this WRITE operation may be serialized by persistent reserve from a distributed client. When this keyword is specified, z/OS I/O operations are permitted to the devices while the persistent reserve is held. This keyword should only be used when the invoking program is coordinating I/O activity between z/OS and the distributed client that owns the persistent reserve. The default is NONE.

To code: Specify a value.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value is left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,REUSECP=NO
,REUSECP=YES
When IOTOKEN=*iotoken* and READ are specified, REUSECP is an optional parameter indicating that the channel program for this operation is exactly the same as the previous channel program. On the first READ request, this keyword is ignored. On subsequent requests, this keyword indicates whether the exact same storage buffers and blocks on the disks are involved in the I/O operations. The exact same channel program is used to perform the I/O. Using this REUSECP=YES may provide some performance advantage since analyzing the extents and storage buffers is not required. The default is REUSECP=NO.

,REUSECP=NO
Indicates that the channel program may vary from invocation to invocation and should be rebuilt on each IOSFBA request.

,REUSECP=YES
Indicates that the channel program does not vary from invocation to invocation and that IOSFBA is instructed to start the previously built channel program without modification if one has been previously built. If no prior channel program was executed for the input IOTOKEN, a new one is built and saved for the next invocation. When REUSECP=YES is specified,

- You must use storage that is fixed for life of the IOTOKEN. Channel programs are not rebuilt to obtain the latest real storage address.
- You must make the address space non-swappable for the life of the IOTOKEN.

If either of these conditions cannot be met, you must not use REUSECP=YES.

,REUSECP=NO
,REUSECP=YES
When IOTOKEN=*iotoken* and WRITE are specified, REUSECP is an optional parameter that indicates that the channel program for this operation is exactly the same as the previous channel program. On the first WRITE request, this keyword is ignored. On subsequent requests, this keyword indicates whether the exact same storage buffers and blocks on the disks are involved in the I/O operations. The exact same channel program is used to perform the I/O. Using this REUSECP=YES may provide some performance advantage since analyzing the extents and storage buffers is not required. The default is REUSECP=NO.

,REUSECP=NO
Indicates that the channel program may vary from invocation to invocation and should be rebuilt on each IOSFBA request.

,REUSECP=YES
Indicates that the channel program does not vary from invocation to invocation and that IOSFBA is instructed to start the previously built channel program without modification if one has been previously built. If no prior channel program was executed for the input IOTOKEN, a new one is built and saved for the next invocation. When REUSECP=YES is specified,

- You must use storage that is fixed for life of the IOTOKEN. Channel programs are not rebuilt to obtain the latest real storage address.
- You must make the address space non-swappable for the life of the IOTOKEN.

If either of these conditions cannot be met, you must not use REUSECP=YES.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value is left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,UNALLOCATE
A required input parameter that unallocates the devices as specified in the device list mapped by the FBADL DSECT (defined in the IOSDFBA macro).

To code: Specify a value.

,WRITE
A required input parameter that writes information to one or more FBA devices as specified by the device I/O list (DEVIOLIST parameter).

To code: Specify a value.

ABEND codes

None.

Return and reason codes

When the IOSFBA macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

Table 99. Return and reason codes for the IOSFBA macro

Return code	Meaning and action
X'00'	Successful operation. **Reason Code** **Meaning/Action** **X'00'** The operation was successful. **X'01'** (ALLOCATE) The requested number of devices (DEVCOUNT) was not available. However, FBADDL_COUNT contains the number of devices actually allocated for this request, which is greater than or equal to the minimum number of devices required (MINDEVCOUNT). **X'02'** (READ, WRITE) The caller requested that the channel program be reused but failed to pass a valid IOTOKEN, so it is not possible to reuse the channel program.
X'04'	Warning error. **Reason Code** **Meaning/Action** **X'01'** (QUERY) Information for one or more devices could not be obtained. **X'02'** (UNALLOCATE) One or more devices in the DEVLIST could not be unallocated.

Table 99. Return and reason codes for the IOSFBA macro (continued)

Return code	Meaning and action
X'08'	Error in the caller's parameters. **Reason Code** **Meaning/Action** **X'01'** IOSFBA abended during parameter validation. **X'02'** (ALLOCATE) MINDEVCOUNT is greater than DEVCOUNT. **X'03'** (ALLOCATE, QUERY) The device list is not properly built. **X'04'** (READ, WRITE) The number of blocks identified in the READ or WRITE request does not properly equate with the amount of I/O buffers provided. **X'05'** (READ, WRITE) The requested extents to be READ or WRITTEN are not within the acceptable range of extents available on the device. **X'06'** An invalid function was specified for the IOSFBA invocation. **X'07'** (UNALLOCATE, ERASE) The device descriptor list is not properly built. **X'08'** (UNALLOCATE, READ, WRITE, ERASE) The device descriptor entry is not properly built. **X'09'** (UNALLOCATE, READ, WRITE, ERASE) The UCB specified in the device descriptor entry is not allocated. **X'0B'** (READ, WRITE) The buffers specified are not properly sized with the physical block size of the device. Buffer sizes must be multiples of the physical block size of the device. **X'0C'** (READ, WRITE) The device I/O list is not properly built. **X'0D'** (READ, WRITE) The device I/O entry is not properly built. **X'0E'** (READ, WRITE) The extent entry is not properly built. **X'0F'** (READ, WRITE) REUSECP=YES was specified, but the address space is swappable. The address space must be non-swappable when REUSEP=YES is specified.

Table 99. Return and reason codes for the IOSFBA macro (continued)

Return code	Meaning and action
X'0C'	Environmental error. **Reason Code** **Meaning/Action** **X'01'** (ALLOCATE) Not enough devices were available to satisfy the requested allocation. Devices must be online, usable, and not already allocated in order to be allocated by this service. **X'02'** (ALLOCATE) Not enough devices provided I/O responses that enabled IOSFBA to validate that they were usable, so IOSFBA was not able to satisfy the requested allocation. Devices must be online, usable, not already allocated, and must properly respond to I/O commands that query device information in order to be allocated by this service. **X'03'** (ALLOCATE, READ, WRITE) IOSFBA detected a serialization problem with one of the devices being allocated. A SYMREC record was written. Consider varying the device offline and investigate prior device usage before attempting to use the identified device again. **X'04'** IOSFBA service is not available. **X'05'** IOSFBA detected that the caller is not in task mode. IOSFBA must be invoked in task mode. **X'06'** IOSFBA detected that the caller is not enabled for I/O interrupts.
X'10'	The READ or WRITE operation was not successful. Check the status blocks for information on the failed I/Os. Note that this return code is only valid for synchronous READ and WRITE operations. Asynchronous READ and WRITE operations are notified by a post code of X'10'.
X'14'	The invoker is not authorized to use this programming service. **Reason Code** **Meaning/Action** **X'01'** The invoker must be in PSW key 0-7 and in supervisor state.

Chapter 109. IOSHXBLK — Request to suspend and resume Basic HyperSwap services

Description

The IOSHXBLK macro is used by authorized applications to request Basic HyperSwap to temporarily suspend its activities. The macro should subsequently be used to allow Basic HyperSwap to resume its activities.

Note that obtaining and holding the block of Basic HyperSwap for a long period of time may cause one of Basic HyperSwap tasks to issue diagnostic abend 2E0.

Environment

The requirements for the caller of IOSHXBLK are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state. Any key.
Dispatchable unit mode:	Task mode
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller may not hold any lock.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IOSHXBLK macro, the caller does not have to place any information into any registers unless using it in register notation for a particular parameter or using it as a base register.

Register
Contents

0
Undefined

1
Used by the service

2-13
Undefined

14-15
Used by the service

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system

1
Unpredictable

2-13
Unchanged

14
Used as a work register by the system

15
Return code

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IOSHXBLK macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSHXBLK.
IOSHXBLK	
ƀ	One or more blanks must follow IOSHXBLK.
BLOCK	Suspend Basic HyperSwap activities.
,NAMEPTR=*nameptr*	*nameptr:* RS-type address or address in register (2) - (12)
,TOKENPTR=*tokenptr*	*tokenptr:* RS-type address or address in register (2) - (12)
UNBLOCK	Resume Basic HyperSwap activities.
,NAMEPTR=*nameptr*	*nameptr:* RS-type address or address in register (2) - (12)

Syntax	Description
,TOKENPTR=*tokenptr*	*tokenptr:* RS-type address or address in register (2) - (12)
,FORCE=NO	**Default**: FORCE=NO
,FORCE=YES	
TESTBLOCK	Test whether all systems in the SYSPLEX support programmatic blocking.
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=*plistver*	
,PLISTVER=IMPLIED_VERSION	**Default**: IMPLIED_VERSION
,MF=S	**Default:** MF=S
,MF=(L,*mfctrl*,*mfattr*,0D)	
,MF=(E,*mfctrl*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSHXBLK macro invocation. The name must conform to the rules for an ordinary assembler language symbol. The default is no name.

BLOCK, UNBLOCK and TESTBLOCK are mutually exclusive keys. This set is required, but only one key may be specified.

BLOCK
Suspend Basic HyperSwap activities.

,NAMEPTR=*nameptr*
The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-character area that contains the name of the application issuing the BLOCK request.

,TOKENPTR=*tokenptr*
The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-byte area that contains a token uniquely identifying this instance of a BLOCK request.

UNBLOCK
Resume Basic HyperSwap activities.

,NAMEPTR=*nameptr*
The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-character area that contains the name of the application issuing the UNBLOCK

request. If this name does not match the name of the application for which Basic HyperSwap is currently blocked, the UNBLOCK request will be rejected.

,TOKENPTR=*tokenptr*
The name (RS-type), or address in register (2)-(12), of a required 4-byte input that contains the address of a 8-byte area that contains the token specified on the previous BLOCK request. If this token does not match the token specified on the BLOCK request for which Basic HyperSwap is currently blocked or if FORCE=YES is not specified on this UNBLOCK request, the request will be rejected.

,FORCE=YES
,FORCE=NO
An optional keyword input that specifies whether to allow unblock if the token value on an UNBLOCK request does not match the token value on the BLOCK request for which Basic HyperSwap is currently blocked. The default is NO.

,FORCE=YES
Even though the token specified on this request does not match the token specified on the BLOCK request for which Basic HyperSwap is currently blocked, honor this UNBLOCK request as long as the name specified on this request matches the name specified on the BLOCK request for which Basic HyperSwap is currently blocked.

,FORCE=NO
Do not unblock unless name and token specified on this UNBLOCK request match the name and token specified on the BLOCK request for which Basic HyperSwap is currently blocked.

TESTBLOCK
Test whether all systems in the SYSPLEX support programmatic blocking.

End of mutually exclusive required keys.

,RETCODE=*retcode*
The name (RS-type) of an optional full-word output variable, or register (2)-(12) or (15), into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

,RSNCODE=*rsncode*
The name (RS-type) of an optional full-word output variable, or register (2)-(12), into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

,PLISTVER=*plistver*
,PLISTVER=IMPLIED_VERSION
An optional byte input decimal value in the "1-1" range that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number, and the parameter list will be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

,MF=S
,MF=(L,*mfctrl*, *mfattr* | 0D)
,MF=(E,*mfctrl*,COMPLETE)
An optional keyword input which specifies the macro form. The default is S.

,MF=S
Specifies the standard form of the macro. The 'S' form generates code to put the parameters into an in-line parameter list and invoke the desired service. Full checking for required macro keys is done along with supplying defaults for omitted optional parameters.

,MF=(L,*mfctrl* ,*mfattr* | 0D)
Specifies the list form of the macro. The 'L' form defines an area to be used for the parameter list. Only the PLISTVER key may be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

,mfctrl
A required input. It is the name of a storage area for the parameter list.

***,mfattr* | 0D**
An optional 60-character input string that varies from 1 to 60 characters. Use it to force boundary alignment of the parameter list. Use only 0F or 0D. The default is 0D, which forces the parameter list to a doubleword boundary.

,MF=(E,*mfctrl*,COMPLETE)
Specifies the execute form of the macro. The 'E' form generates code to put the parameters into the parameter list specified by *mfctrl* and provides full syntax checking with default setting.

,mfctrl
A required input. It is the name (RS-type), or address in register (1)-(12), of a storage area for the parameter list.

,COMPLETE
An optional keyword input which specifies the degree of macro parameter syntax checking. The default is COMPLETE.

Checking for required macro keys is done and defaults are supplied for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IOSHXBLK macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

Table 100. Return Codes for the IOSHXBLK Macro

Hexadecimal return code	Reason codes, meaning and action
00	Basic HyperSwap services have successfully completed the requested action. For TESTBLOCK requests, all systems support programmatic blocking.

Table 100. Return Codes for the IOSHXBLK Macro (continued)

Hexadecimal return code	Reason codes, meaning and action
04	Basic HyperSwap services did not complete the requested action. For TESTBLOCK requests, at least one system does not support programmatic blocking. **Reason code** **Meaning/Action** **00** Basic HyperSwap services not started. **01** Basic HyperSwap services initialization incomplete. **41** HyperSwap in progress. **42** Basic HyperSwap services are temporarily busy. **43** Basic HyperSwap blocking services are in unknown state due to a system error. Unblock with Force option is required to clear this condition. **44** IOSHXBLK service timed out waiting for Basic HyperSwap blocking services to complete the request. Unblock with Force option may be required before retrying the request.

Table 100. Return Codes for the IOSHXBLK Macro (continued)

Hexadecimal return code	Reason codes, meaning and action
08	Basic HyperSwap services did not accept the request. **Reason code** **Meaning/Action** **01** Invalid function code. **02** Caller is not in task mode. **03** Caller is not in supervisor state. **04** Caller is not in primary ASC mode. **05** Caller is not enabled for I/O and external interrupts. **06** Caller is holding lock(s). **07** System resources unavailable to copy request. **08** Internal function code mismatch. **09** Routing of request in SYSPLEX failed. **0A** System service error occurred while processing the request. **0B** Blocking not supported by Basic HyperSwap master system. **0C** Blocking not supported by one or more Basic HyperSwap member systems. **0D** HyperSwap API services address space is not active on one or more Basic HyperSwap member systems.

Table 100. Return Codes for the IOSHXBLK Macro (continued)

Hexadecimal return code	Reason codes, meaning and action
0C	Programming error. **Reason code** **Meaning/Action** **01** BLOCK request while already blocked. The name pointed to by NAMEPTR on the BLOCK request is the same as the name of the application currently blocking Basic HyperSwap. **02** BLOCK request while already blocked. The name pointed to by NAMEPTR on the BLOCK request is different from the name of the application currently blocking Basic HyperSwap. **03** UNBLOCK request while not blocked. **04** Name or token specified on an UNBLOCK request did not match name or token specified on the BLOCK request that is currently blocking Basic HyperSwap. **05** Not a BLOCK, UNBLOCK or TESTBLOCK request.
10	Environmental error. **Reason code** **Meaning/Action** **00** Caller not authorized or invocation environment wrong. **01** Unable to establish a recovery environment. **02** System resources unavailable for the request. **03** ALESERV service failed for the request.
20	System abend occurred while processing request.

Chapter 110. IOSINFO — Obtain the subchannel number for a UCB

Description

The IOSINFO macro obtains the subchannel number for a specified unit control block (UCB). The macro returns the subsystem identification word (SID), which identifies the subchannel number of the UCB, in a user-specified location. The SID is a fullword value whose first halfword contains X'0001' and ending halfword contains the subchannel number.

Environment

The issuer of IOSINFO must be executing:

- In 31-bit addressing mode
- In either task mode or SRB mode
- Locked or unlocked

Additionally, the issuing program must include the CVT and IHAPSA mapping macros. All addresses must be 31-bit addresses and the issuing program must pass a below 16 megabyte actual or captured UCB.

Input register information

Before entry to this macro, register 13 must contain the address of a standard 18-word save area.

Output register information

After the caller issues the macro, the macro might use some registers as work registers or might change the contents of some registers. When the macro returns control to the caller, the contents of these registers are not the same as they were before the macro was issued. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the macro

1
Contains the SID if the return code in register 15 is 0; otherwise, used as a work register by the macro.

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Syntax

The IOSINFO macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSINFO.
IOSINFO	
ƀ	One or more blanks must follow IOSINFO.
FUNCTN=SUBCHNO	
,UCB=*ucb addr*	*ucb addr*: A-type address or register (0) - (15).
,OUTPUT=*output addr*	*output addr*: A-type address or register (0) - (14).
,RTNCODE=*retcde addr*	*retcde addr*: A-type address or register (0) - (15).

Parameters

The parameters are explained as follows:

FUNCTN=SUBCHNO
Specifies that a subchannel number is to be obtained.

,UCB=*ucb addr*
Specifies the address of a fullword on a fullword boundary containing the address of a unit control block (UCB).

,OUTPUT=*output addr*
Specifies the address of a fullword on a fullword boundary that will contain the subsystem identification word (SID) upon completion.

The SID is a fullword value that identifies the subchannel. The first halfword is X‘0001’, and the last halfword contains the subchannel number.

The output address must reside in 31-bit addressable storage.

,RTNCODE=*retcde addr*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. The specified storage location must be a fullword on a fullword boundary.

The return code address must reside in 31-bit addressable storage.

Return codes

When control returns from IOSINFO, GPR 15 (and *retcde addr*, if you coded RTNCODE) contains one of the following return codes:

Hexadecimal Code	Meaning
00	The address specified on the OUTPUT parameter contains the SID.*
04	The UCB was disassociated from the subchannel at the time of the IOSINFO service routine invocation.

* In some cases, the subchannel number in the SID might not be valid. Any disassociation of the UCB and the subchannel means the subchannel number in the SID is not valid. If the UCB is disassociated from the subchannel after the IOSINFO service routine invocation, no notification can be given.

Example 1

Obtain the subchannel number for a UCB whose address is in register 1. Specify the SID output to be placed in register 2 and the return code to be placed in register 3.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=(1),OUTPUT=(2),RTNCODE=(3)
```

Example 2

Obtain the subchannel number for a UCB whose address is in location ADDR. Specify the SID output to be placed in location ADDX and the return code to be placed in register 3.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=ADDR,OUTPUT=ADDX,RTNCODE=(3)
```

Example 3

Obtain the subchannel number for a UCB whose address is in register 2. Specify the SID output to be placed in register 3 and the return code to be placed in location ADDR.

```
IOSINFO  FUNCTN=SUBCHNO,UCB=(2),OUTPUT=(3),RTNCODE=ADDR
```

Chapter 111. IOSLOOK — Locate unit control block

Programming note: The IOSLOOK macro is a deprecated programming interface. Use the UCBLOOK macro instead.

Description

The IOSLOOK macro locates the unit control block (UCB) associated with a device number. To use IOSLOOK, you must be executing in supervisor state. Register 13 must point to a 16-word save area where the macro stores registers 0 through 15 at offset 0. You must also include a DSECT for both the CVT (using the CVT mapping macro) and the IOCOM (using the IECDIOCM mapping macro).

Syntax

The IOSLOOK macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede IOSLOOK.
IOSLOOK	
␣	One or more blanks must follow IOSLOOK.
DEV=(*reg*)	*reg*: Register (0) - (12), (14), (15).
	Default: DEV=(6).
,UCB=(*reg*)	*reg*: Register (0) - (12).
	Default: UCB=(7).

Parameters

The parameters are explained as follows:

DEV=(*reg*)
Specifies a general purpose register, symbolic or absolute, that contains the hexadecimal device number, right justified. If this parameter is omitted, register 6 is assumed.

,UCB=(*reg*)
Specifies a general purpose register, symbolic or absolute, that will be used to return the address of the UCB common segment. If this parameter is omitted, register 7 is assumed. If the UCB address cannot be found, then the contents of this register are unpredictable.

Note: The UCB must reside in 24-bit addressable storage.

Return codes

When IOSLOOK macro returns control to your program, GPR 15 contains a return code.

Table 101. Return Codes for the IOSLOOK Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: UCB address was found.
04	**Meaning**: Device number was invalid or no UCB exists.

Example

Find the UCB address for device 250. Register 2 contains the value X'00000250'. The UCB address is to be returned in register 5 and UCBPTR is equated to 5.

```
IOSLOOK DEV=(2),UCB=(UCBPTR)
```

Chapter 112. IOSODS — IOS offline device service

Description

The IOS Offline Device Service macro provides the interface for authorized code to mark a device offline and in use by a system component.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and key 0.
Dispatchable unit mode:	Task mode.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	None

Programming requirements

None.

Restrictions

- The caller may not hold any locks.
- The caller is required to pin the UCB for the device before invoking the IOSODS macro service. Pinning the UCB will insure that the proper identification of the user of the device will be displayed if the installation should try the dynamically delete it.
- Issuers of the IOSODS macro service must provide recovery and resource termination managers to insure that the device is freed for use by other applications in case of an unexpected failure or cancellation of the address space.

Input register information

None.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1–14
Unchanged

15
Return code

.

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Performance implications

None.

Syntax

The standard form of the IOSODS macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IOSODS.
IOSODS	
␣	One or more blanks must follow IOSODS.
ON	**Default**: None.
,BYPONLINEFENCE	Requests to bypass the fence that prevents a device from successfully transitioning into the IOSODS ON state. **Default**: None.
,BYPALIAS={NO|YES}	**Default:** NO
,WLMPAVSUSPEND	This function is disabled.
OFF	**Default**: None.
,WLMPAVRESTORE	This function is disabled.
,DEVN=*devn*	*devn*: RS-type name or address in register (2) - (12).
,DEVNCHAR=*devnchar*	*devnchar*: RS-type address or register (2) - (12).
SCHSET={*schset*|0}	**Default**: 0

Syntax	Description
,LDEVNCHAR=*ldevnchar*	*ldevnchar*: RS-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type name or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type name or register (2) - (12).
,PLISTVER={*plistver* | MAX | IMPLIED_VERSION}	**Default**: IMPLIED_VERSION

Parameters

The parameters are explained as follows:

The following is a set of mutually exclusive keywords. This set is required; only one keyword must be specified.

ON
Keyword that indicates the input device number is to be marked as offline and in use by a system component.

,BYPONLINEFENCE
Optional keyword that indicates that the device should be allowed to have dynamic pathing established, even if the device is fenced for VARY establishing dynamic pathing. When this keyword is specified, z/OS I/O operations will be permitted to bypass a fence, if established, that would have prevented the device from having dynamic pathing established. Invokers of this service should understand why a device may have been fenced for dynamic pathing, and understand why it is okay to bypass this type of fence before using this keyword.

Default: None.

,BYPALIAS=NO
,BYPALIAS=YES
Optional keyword that indicates whether the device initialization is to include HyperPAV alias initialization.

BYPALIAS=NO
HyperPAV alias initialization will be performed, if necessary.

BYPALIAS=YES
HyperPAV alias initialization will not be performed.

Default: NO

,WLMPAVSUSPEND
Optional keyword that indicates that the Work Load Manager dynamic alias tuning capability for the device (if applicable) will be suspended. Note that it is up to the user to restore this capability through an IOSODS OFF request with the WLMPavRestore keyword specified. This function is currently disabled.

Default: None.

,UNCOND
Requests that parallel access volumes be initialized.

Default: None.

ON

,UNCOND
Optional keyword that indicates to unconditionally activate some of the system and device dependent features. When UNCOND is used in conjunction with BYPALIAS(NO), the parallel access volumes will be initialized. Initialization processing will be requested even when the device is already in use by a system component (for example, a prior IOSODS ON function).

Default: None.

OFF
Keyword that indicates the input device number is no longer to be marked as offline and in use by a system component.

,WLMPAVRESTORE
Optional keyword that indicates that the Work Load Manager dynamic alias tuning capability for the device (if applicable) will be restored. Note that it is up to the user to restore this capability through an IOSODS ON request with the WLMPavSuspend keyword specified. This function is currently disabled.

Default: None.

This ends the of set of mutually exclusive required keywords.

,DEVN=*devn*
The name (RS-type), or address in register (2) - (12), of a required halfword input that specifies the device number in binary of the device that is to be operated on.

,DEVNCHAR=*devnchar*
The name (RS-type), or address in register (2) - (12), of an optional byte input that specifies the device number in EBCDIC, of the device that is to be operated on.

,SCHSET=0
,SCHSET=*schset*
The name (RS-type), or address in register (2) - (12), of an optional byte input that specifies the subchannel set of the device.

Default: 0

,LDEVNCHAR=*ldevnchar*
The name (RS-type), or address in register (2) - (12), of a 5-character input that specifies the logical device number, in EBCDIC, of the device whose UCB address is to be obtained.

Note: A logical device number is represented by a 1-digit subchannel set id followed by the 4-digit device number, sdddd.

,RETCODE=*retcode*
The name (RS-type), or register (2) - (12), of an optional fullword output into which the return code is to be copied from GPR 15.

,RSNCODE=*rsncode*
The name (RS-type), or register (2) - (12), of an optional fullword output into which the reason code is to be copied from GPR 0.

,PLISTVER=*plistver* | MAX | IMPLIED_VERSION
An optional byte input decimal value in the "1–1" range that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number and will cause the parameter list to be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as this will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

Default: IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

ABEND codes

None.

Return codes

Return and reason codes, in hexadecimal, from the IOSODS macro are as follows:

Hexadecimal return code	Hexadecimal reason code	Meaning
00	None	The requested function executed successfully.
08	01	The requested function failed because the input device number was not found.
08	02	A request to allocate an offline device was made, but it is already in use by a system component (ON function).
08	03	A request to unallocate a device was made, but the device is not currently in use (OFF function).
08	04	IOS Path Validation failed (ON function).
08	05	IOS Dynamic Pathing function failed (OFF function).
08	06	Allocation service to set UCBNALOC failed (ON function).
20	None	An unexpected error occurred. The recovery routine recovered and returned control to the caller.

IOSODS - List form

Use the list form of the IOSODS macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

Syntax

The list form of the IOSODS macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IOSODS.
IOSODS	
␣	One or more blanks must follow IOSODS.
,PLISTVER={*xplistver* | MAX | IMPLIED_VERSION}	**Default**: IMPLIED_VERSION

Syntax	Description
,MF=(L,*xmfctrl*{,*xmfattr*|0D})	*xmfctrl*: RS-type name or address in register (2) - (12). *xmfattr*: Any text up to 60 characters. **Default**: 0D

Parameters

The parameters are explained under the standard form of the IOSODS macro, with the following exception:

,MF=(L,*xmfctrl*{,*xmfattr*|0D})
L specifies the list form of the macro. The list form defines an area to be used for the parameter list. Only the PLISTVER key may be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

,xmfctrl
This required input is the RS-type name, or address in register (1) - (12), of a storage area for the parameter list.

***,xmfattr*|0D**
This is an optional 60 character input string which is used to force boundary alignment of the parameter list. Use only 0F or 0D.

Default: 0D which forces the parameter list to a doubleword boundary.

IOSODS - Execute form

Use the execute form of the IOSODS macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSODS macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IOSODS.
IOSODS	
␣	One or more blanks must follow IOSODS.
ON	**Default**: None.
,BYPONLINEFENCE	Requests to bypass the fence that prevents a device from successfully transitioning into the IOSODS ON state. **Default**: None.

Syntax	Description
,BYPALIAS={NO|YES}	**Default:** NO
,WLMPAVSUSPEND	This function is disabled.
OFF	**Default**: None.
,WLMPAVRESTORE	This function is disabled.
,DEVN=*devn*	*devn*: RS-type name or address in register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type name or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type name or register (2) - (12).
,PLISTVER={*xplistver* | MAX | IMPLIED_VERSION}	**Default**: IMPLIED_VERSION
,MF=(E,*xmfctrl*[,COMPLETE])	*xmfctrl*: RS-type name or address in register (2) - (12). **Default**: COMPLETE

Parameters

The parameters are explained under the standard form of the IOSODS macro, with the following exception:

,MF=(E,*xmfctrl*[,COMPLETE])
E specifies the execute form of the macro. The execute form generates code to put the parameters into the parameter list specified by *xmfctrl* and provides full syntax checking with default setting.

,xmfctrl
This required input is the RS-type name, or address in register (1) - (12), of a storage area for the parameter list.

,COMPLETE
An optional keyword which specifies the degree of macro parameter syntax checking. When complete checking is enabled, required parameters are checked and defaults are supplied for omitted optional parameters.

Default: COMPLETE

Chapter 113. IOSPTHV — Validate I/O paths

Description

The IOSPTHV macro enables authorized callers to validate the physical connectivity and availability of a channel path to a device. A path is considered available if an I/O operation can be initiated down a path, and the device can be selected. Validation does not guarantee that the device and path are error free. The IOSPTHV function depends on the availability of the IOS Address Space (IOSAS). IOSAS is started after Master Scheduler Initialization (MSI), and may be unavailable for periods of time during recovery. The issuer of the IOSPTHV macro must be able to handle the return/reason code indicating that the IOSAS is not active.

IOSPTHV is similar to UCBINFO PATHINFO and the VARY command, but there are important differences. UCBINFO returns status based on UCB indicators that might be outdated. Unlike the VARY command, IOSPTHV does not change UCB path status indicators or dynamically vary paths online or offline. IOSPTHV only tests physical connectivity. Examples of when you might validate a path include:

- To verify the current availability of a specific path to a specific device and present any path-related errors to a user.
- As a first step in diagnosing path-related problems.
- After installing a device, to verify channel to device connectivity before issuing a VARY command for the device.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum Authorization:	Supervisor state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space

Programming requirements

None.

Restrictions

Do not have any enabled, unlocked task (EUT) FRRs established. If issued during IPL before the IOSAS (IOS address space) has initialized, MSI must have completed and WAIT=YES must be specified on the IOSPTHV macro.

If you attempt to validate a path to an active teleprocessing device (device types 2701, 2702, and 2703) or to an OSA or CTC device in use by VTAM with a long running I/O active, you will receive an error return and reason code.

Input register information

Before issuing the IOSPTHV macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller of the IOSPTHV macro, the access registers (ARs) contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after invoking a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IOSPTHV macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSPTHV.
IOSPTHV	
ƀ	One or more blanks must follow IOSPTHV.
,DEVN=*device number*	*device number*: RX-type address or address in register (2) - (12).

Syntax	Description
,CHPID=*path id*	*path id*: RX-type address or address in register (2) - (12).
,MSGBUF=*msgbuf addr*	*msgbuf addr*: RX-type address or address in register (2) - (12).
	Default: none
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or address in register (2) - (12).
	Default: none
,TIME=*time*	*time*: RX-type address or address in register (2) - (12).
	Default: 5 seconds
,RETCODE=*return code*	*return code*: RX-type address or address in register (2) - (12).
,RSNCODE=*return code*	*reason code*: RX-type address or address in register (2) - (12).
,WAIT=*NO*	**Default:** NO
,WAIT=*YES*	

Parameter descriptions

The parameters are explained as follows:

,DEVN=*device number*
Specifies the device's binary device number (0000 - FFFF). IOSPTHV checks the availability of the path you specify on the CHPID parameter to the device you specify on DEVN. IOSPTHV pins the device so that the device's UCB and other related data structures are not dynamically deleted while IOSPTHV is validating the path. When IOSPTHV completes processing, it unpins the device.

,CHPID=*path id*
Specifies the ID (00 - FF) of the channel path that IOSPTHV validates for physical availability. To determine the ID for a specific channel path connected to the device specified on DEVN, use the UCBINFO PATHINFO macro or the DISPLAY MATRIX operator command.

,MSGBUF=*msgbuf addr*
Specifies the address of a 71-character area into which IOSPTHV is to place diagnosis information. IOSPTHV uses this buffer only if the return code is X'04' and the reason code is X'04'. This information consists of the same message that is issued by the VARY command for comparable errors. This message is the last message that MVS would have issued if a VARY PATH command had been issued and a similar error had been encountered. IOSPTHV does not issue a message to the operator console.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character area that contains the MVS I/O configuration token that you supply to IOSPTHV. You can obtain this token by issuing the IOCINFO macro, which is described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*. If the I/O configuration token that is current when IOSPTHV is invoked does not match the token you supply, you are notified through a return code.

If you set the input IOCTOKEN to binary zeros, IOSPTHV sets IOCTOKEN to the current I/O configuration token.

For information about how you can use the configuration token to detect configuration changes, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

,WAIT=NO
,WAIT=YES
Is an optional keyword input that indicates to allow the request to wait for the IOS address space (IOSAS) to initialize or restart (if terminated) before continuing. WAIT=NO is the default.

WAIT=NO: Only process if the IOS address space has been initialized and not terminated.

WAIT=YES: Allows the request to wait for the IOSAS space to initialize as long as MSI has completed. Allows the request to wait for the IOS address space to reinitialize if terminated. The user of this keyword must ensure that no no resources are held that can cause the IOSAS not to initialize.

,TIME=*time*
Specifies an 8-byte field containing the maximum amount of time, in seconds, that IOSPTHV can run before being purged. The default for the TIME parameter is 5 seconds.

The time interval, whose address resides in virtual storage, is presented as zoned decimal digits of the form:

HHMMSSth, where:

HH
is hours (24-hour clock)

MM
is minutes

SS
is seconds

t
is tenths of seconds

h
is hundredths of seconds

IOSPTHV runs until one of the following occurs:

- IOSPTHV completes successfully or unsuccessfully
- The interval that you specify on the TIME=parameter expires
- The MIH interval for the device expires.

Note that the TIME parameter allows you to set an expiration time that is specific to IOSPTHV. The MIH interval, however, is used by other services associated with the device. Using the TIME parameter allows you to set an expiration time that is shorter than the MIH interval or the time that it takes the IOSPTHV macro to complete successfully.

,RETCODE=*rc*
Specifies the location or register where the system is to place the return code. The system copies the return code into the location from register 15.

,RSNCODE=*rsncode*
Specifies the location or register where the system is to place the reason code. The system copies the reason code into the location from register 0.

Return and reason codes

Return codes, in hexadecimal, from the IOSPTHV macro are as follows:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: IOSPTHV processing completed successfully. IOSPTHV successfully validated the specified path. The path is physically available. **Action**: None
04	04	**Meaning**: IOSPTHV did not successfully validate the specified path because the path was not physically available. **Action**: You need to investigate the problem further. Trying to vary the path online may produce further diagnosis data.
04	08	**Meaning**: User-specified time interval on the TIME keyword expired before the I/O completed. **Action**: Verify that the time interval was long enough. Note that this return code is issued only if the time expired before the MIH interval. You can use the D MIH command or the MIHQUERY macro to determine the MIH interval for the device.
08	04	**Meaning**: IOSPTHV did not successfully validate the specified path because the device number specified on the DEVN keyword is not valid. **Action**: Ensure that you specified the device number correctly and retry the operation. Use the IOCTOKEN keyword to ensure that the UCB for that device number was not dynamically changed or deleted.
08	08	**Meaning**: IOSPTHV did not successfully validate the specified path because the path specified on the CHPID keyword is not valid. **Action**: Verify your program to ensure that the correct CHPID was passed and retry the operation. Use the IOCTOKEN keyword to ensure that the CHPID for the device was not dynamically changed or deleted.
08	0C	**Meaning**: IOSPTHV did not successfully validate the specified path because the time specified on the TIME keyword was not valid. **Action**: Ensure that the time specified contains valid zoned decimal digits in the proper range.
08	20	**Meaning**: IOSPTHV did not successfully validate the specified path because the UCB definition for the device represented by the look-up argument (device number) has changed and is no longer consistent with the UCB definition represented by the input I/O configuration token. (This return code is only valid for callers using the IOCTOKEN keyword.) **Action**: Ensure that the device number and CHPID are still valid and retry the operation passing a current IOCTOKEN.
08	24	**Meaning**: Processing cannot be performed before the IOS address space (IOSAS) has initialized (unless MSI has has completed and the WAIT=YES keyword was specified). **Action**: Retry the operation later in the IPL, after the IOSAS has been initialized, or after MSI by using the WAIT=YES keyword.
08	28	**Meaning**: IOSPTHV did not successfully validate the specified path because an ESTAE environment could not be established. **Action**: Ensure that sufficient private area storage exists and retry the operation.
0C	None	**Meaning**: An unexpected error occurred. **Action**: Record the return code and supply it to the appropriate IBM support personnel.

Example

Determine if a channel path to the SYSRES device is available without changing the online/offline status of the path. Scan through all UCBs, using the UCBSCAN macro, and put copies of the DASD UCBs the program finds in a user-supplied work area called UCBSTOR. When the program finds the SYSRES device, issue the UCBINFO macro to obtain information about the device path and type of channel path for the specified UCB. Information, such as the channel path ID and online status, will appear in the IOSDPATH data area. The program looks through the channel path information until it finds an online path. Issue the IOSPTHV macro to test whether the online path is available.

```
IOSSPTHV CSECT
IOSSPTHV AMODE 31                      31-BIT ADDRESSING MODE
IOSSPTHV RMODE ANY                     Rmode any
         SPACE 1
*.....................................................................*
*         REGISTER ASSIGNMENTS                                        *
*.....................................................................*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6                     Dynamic area register
UCBPTR7  EQU   7                     UCB Pointer
R8       EQU   8
R9       EQU   9                     Module base register
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13                    Pointer to standard save area
R14      EQU   14
R15      EQU   15
         SPACE 3
         TITLE 'IOSSPTHV -  IOSPTHV Sample Program'
*.....................................................................*
*                                                                     *
*        Standard Entry Linkage                                       *
*                                                                     *
*.....................................................................*
         PRINT GEN
         USING *,R9              Sets up base register
ENTRY    STM   R14,R12,12(R13)   Save caller's registers
         LR    R9,R15            Establish module base register
         MODESET KEY=ZERO,MODE=SUP
         LA    R0,DYNSIZE        Load length of dynamic area
         STORAGE OBTAIN,LENGTH=((R0)),SP=233 Gets dynamic area
         LR    R6,R1             Gets dynamic area address
         USING DYNAREA,R6        Sets up dynamic area
         ST    R13,SAVE+4        Save caller's save area address
         LA    R15,SAVE          Get this module's save area address
         ST    R15,8(R13)        Save this modules save area address
*                                in caller's save area.
         LR    R13,R15           Set up addressability to this
*                                module's save area.
         B     MAINLINE
         DC    CL8'IOSSPTHV'
         DC    CL8'&SYSDATE'
         DC    CL8'&SYSTIME'
         TITLE 'IOSSPTHV - SPTHV mainline '
*.....................................................................*
*                                                                     *
*        MAINLINE                                                     *
*                                                                     *
*.....................................................................*
MAINLINE DS   0H
*
```

```
             L     10,X'10'          Load CVT pointer
             USING CVT,10
             TM   CVTDCB,CVTOSEXT    Is the OSLEVEL extension present
             BNO  NO_IOSPTHV         No, pre-MVS/SP Version 3 system
*
             TM   CVTOSLV1,CVTH5510  Running on version HBB5510?
             BNO  NO_IOSPTHV         No, pre-HBB5510 system. IOSPTHV
*                                    supported on HBB5510 and above
*.....................................................................*
```

```
*                                                                     *
*   Set up addressability to a storage area called UCBSTOR into which *
*   the UCBSCAN macro will return the UCBs of devices it locates.     *
*                                                                     *
*.....................................................................*
             LA       UCBPTR7,UCBSTOR    Get address of work area
             USING    UCB,UCBPTR7        Set up addressability
*
*.....................................................................*
*                                                                     *
*   Clear the UCBSCAN work area.                                      *
*                                                                     *
*.....................................................................*
             LA  R0,SCANWORK             Set storage address
             LA  R1,100                  Set storage length
             SR  R15,R15                 Clear second operand
             MVCL  R0,R14                Clear the storage
*
*.....................................................................*
*                                                                     *
*   Loop through all DASD UCBs looking for the SYSRES volume.         *
*                                                                     *
*   Note: There must be a SYSRES volume, and hence it will be found   *
*         in the scan loop which follows.                             *
*                                                                     *
*.....................................................................*
SCANLOOP UCBSCAN COPY,                                                X
               WORKAREA=SCANWORK,                                     X
               UCBAREA=UCBSTOR,                                       X
               DEVCLASS=DASD,                                         X
               MF=(E,SCANLIST)
*.....................................................................*
*                                                                     *
*   If UCBSCAN returned a UCB, check whether the UCB represents       *
*   the SYSRES volume.  If it isn't, continue checking more UCBs.  If *
*   the UCB represents the SYSRES device, end the loop.               *
*                                                                     *
*.....................................................................*
             LTR R15,R15            Test return code
             BNZ EXIT_ERROR         Exit if non-zero
             TM UCBSTAT,UCBSYSR     Test if SYSRES volume
             BZ SCANLOOP            Keep looping if not
```

```
*
*.....................................................................*
*                                                                     *
*   Issue the UCBINFO macro to obtain path-related information.       *
*   UCBINFO returns this information in a field called PATHSTOR,      *
*   mapped by IOSDPATH.                                               *
*                                                                     *
*   Note- Since the device whose path information is sought is the    *
*         SYSRES device, an online path is certain to be found.       *
*         No loop counter is used.                                    *
*                                                                     *
*.....................................................................*
*
         UCBINFO PATHINFO,                                            X
               DEVN=UCBCHAN,                                          X
               PATHAREA=PATHSTOR,                                     X
               MF=(E,INFOLIST)
*.....................................................................*
*                                                                     *
*   If UCBINFO cannot retrieve path-related information, that is, you *
*   receive a non-zero return code, exit program.                     *
*                                                                     *
*.....................................................................*
             LTR      R15,R15          Test for 0 return code
             BNZ      EXIT_ERROR       Exit if bad RC
*.....................................................................*
*                                                                     *
*   Loop through the channel path ID array entries returned in        *
*   PATHSTOR to find the first online path.  An online path           *
*   is represented by a flag in the array.                            *
*                                                                     *
*.....................................................................*
             LA      R10,PATHSTOR      Address of PATHINFO data
             USING   PATH,R10          Set up addressability to
*                                      path information.
             SR      R8,R8             CHPID array index register.
CHPID_LOOP   IC      R11,PATHBITS(R8)  Get flags from array entry.
             STC     R11,PATHSAVE      Save entry
```

```
             TM      PATHSAVE,X'04'    Test if the path is online
             BO      CHPID_EXIT        If so, exit the loop
             LA      R8,L'PATHCHPIDARRAY(R8)   Increment array index
             B       CHPID_LOOP
CHPID_EXIT   LH      R11,PATHCHPID(R8) Get the ID for the online
*                                      channel path.
             STC     R11,CHPID         Save the ID for the online
*                                      channel path.
*.....................................................................*
*                                                                     *
*   Test the availability of the first online path to the SYSRES      *
*   volume by issuing the IOSPTHV macro.  Supply the channel          *
*   path ID of the online path on the CHPID parameter.                *
*                                                                     *
*   Note: Although the logical path mask (LPM) indicated that         *
*         the path was logically online to the device, it is          *
*         possible that the path is not operational.  IOSPTHV         *
*         performs an I/O operation down the path to                  *
*         determine if a non-operational condition exits.             *
*                                                                     *
*.....................................................................*
         IOSPTHV DEVN=UCBCHAN,                                        X
               CHPID=CHPID,                                           X
               MF=(E,PTHVLIST)
```

```
*.....................................................................*
*                                                                     *
*   A zero return code indicates an operational path to               *
*   the specified device.  A non-zero return code indicates           *
*   a non-operational path.  In the latter case, examine the          *
*   return and reason code to determine the cause.                    *
*                                                                     *
*.....................................................................*
             LTR R15,R15
             BZ  PATH_OK
             B PATH_NOK
PATH_OK      DS 0D
          WTO   'IOSSPTHV-FIRST ONLINE PATH TO SYSRES VALIDATED',      X
                ROUTCDE=(11),DESC=(2)
             B EXIT
PATH_NOK     DS 0D
          WTO   'IOSSPTHV-FIRST ONLINE PATH TO SYSRES NOT VALIDATED',  X
                ROUTCDE=(11),DESC=(2)
*
             B EXIT
*.....................................................................*
*                                                                     *
*    Return a message to tell the user that the                       *
*    IOSPTHV macro is not available on the system executing           *
*    this sample program.                                             *
*                                                                     *
*.....................................................................*
NO_IOSPTHV     DS       0H
         WTO   'IOSSPTHV - IOSPTHV SUPPORTED IN HBB5510 AND HIGHER',   X
               ROUTCDE=(11),DESC=(2)
             B EXIT
*.....................................................................*
*                                                                     *
*    Return a message to the user alerting the user to an error       *
*    encountered during execution of this sample program.             *
*                                                                     *
*.....................................................................*
EXIT_ERROR   DS       0H
         WTO   'IOSSPTHV - THE SAMPLE ENCOUNTERED AN ERROR',           X
               ROUTCDE=(11),DESC=(2)
*.....................................................................*
*                                                                     *
*    Clean up and exit.                                               *
*                                                                     *
*.....................................................................*
EXIT           DS    0H
               L     R13,SAVE+4             Reloads caller's save
*                                           area addr into 11
               LA    R0,DYNSIZE             Loads dynamic area size
               STORAGE RELEASE,SP=233,ADDR=(R6),LENGTH=(R0)
               MODESET KEY=NZERO,MODE=PROB
               LM    R14,R12,12(R13)        Loads return regs
               BR    R14                    Returns to caller
*
*
*.....................................................................*
```

```
*                                                                     *
*    DSECTs to map save areas and dynamic area                        *
*                                                                     *
*.....................................................................*
DYNSTART      DS   0H
DYNAREA       DSECT
*        Save area
```

```
SAVE          DS   18F
              DS   0D             Force doubleword alignment
              SPACE 2
*.....................................................................*
*                                                                     *
*  List forms of macros.  The list and execute forms of these macros  *
*  are used because this module is reentrant.                         *
*                                                                     *
*.....................................................................*
LIST_INFOSERV UCBINFO MF=(L,INFOLIST)   List form of UCBINFO
INFOSERV_END  DS   0D
PATHSTOR      DS   CL256                Storage for the PATHAREA
PATHSTOR_END  DS   0D
LIST_PTHVSERV IOSPTHV  MF=(L,PTHVLIST)   List form of IOSPTHV
PTHVSERV_END  DS   0D
LIST_SCANSERV UCBSCAN MF=(L,SCANLIST)   List form of UCBSCAN
SCANSERV_END  DS 0D
SCANWORK      DS CL100                  Scan work area
SCANWORK_END  DS 0D
UCBSTOR       DS CL48                   UCB copy storage
UCBSTOR_END   DS 0D
*.....................................................................*
*                                                                     *
*  Work variables and data structures local to this module            *
*                                                                     *
*.....................................................................*
CHPID         DS C                      CHPID used for IOSCDR invocation
PATHSAVE      DS C                      Work variable for CHPID array
*                                       entries in the PATHAREA.
END_DYN       DS    0D
DYNSIZE       EQU   *-DYNAREA           Calculates Dynamic area
*
*.....................................................................*
*                                                                     *
*  DSECTs                                                             *
*                                                                     *
*.....................................................................*
IOSSPTHV      CSECT
              TITLE 'IOSSPTHV - DSECT MAPPINGS'
         EJECT
         CVT   LIST=YES,DSECT=YES
*
UCB      DSECT
         IEFUCBOB
*
PATHAREA IOSDPATH
         END   IOSSPTHV
```

Chapter 114. IOSSCM — Storage class memory information

Description

The IOSSCM macro retrieves storage class memory (SCM) related information, such as the number of SCM resources and performance statistics.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For `LINKAGE=SYSTEM`, problem state and any PSW key. For `LINKAGE=BRANCH`, supervisor state and key zero.
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit or 64-bit If in AMODE 64, specify `SYSSTATE AMODE64=YES` before invoking this macro.
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	None
Control parameters:	Control parameters must be in the primary address space. For `LINKAGE=BRANCH`, the parameter list (including any data areas pointed to by the parameter list) must reside in fixed or DREF storage.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IOSSCM macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

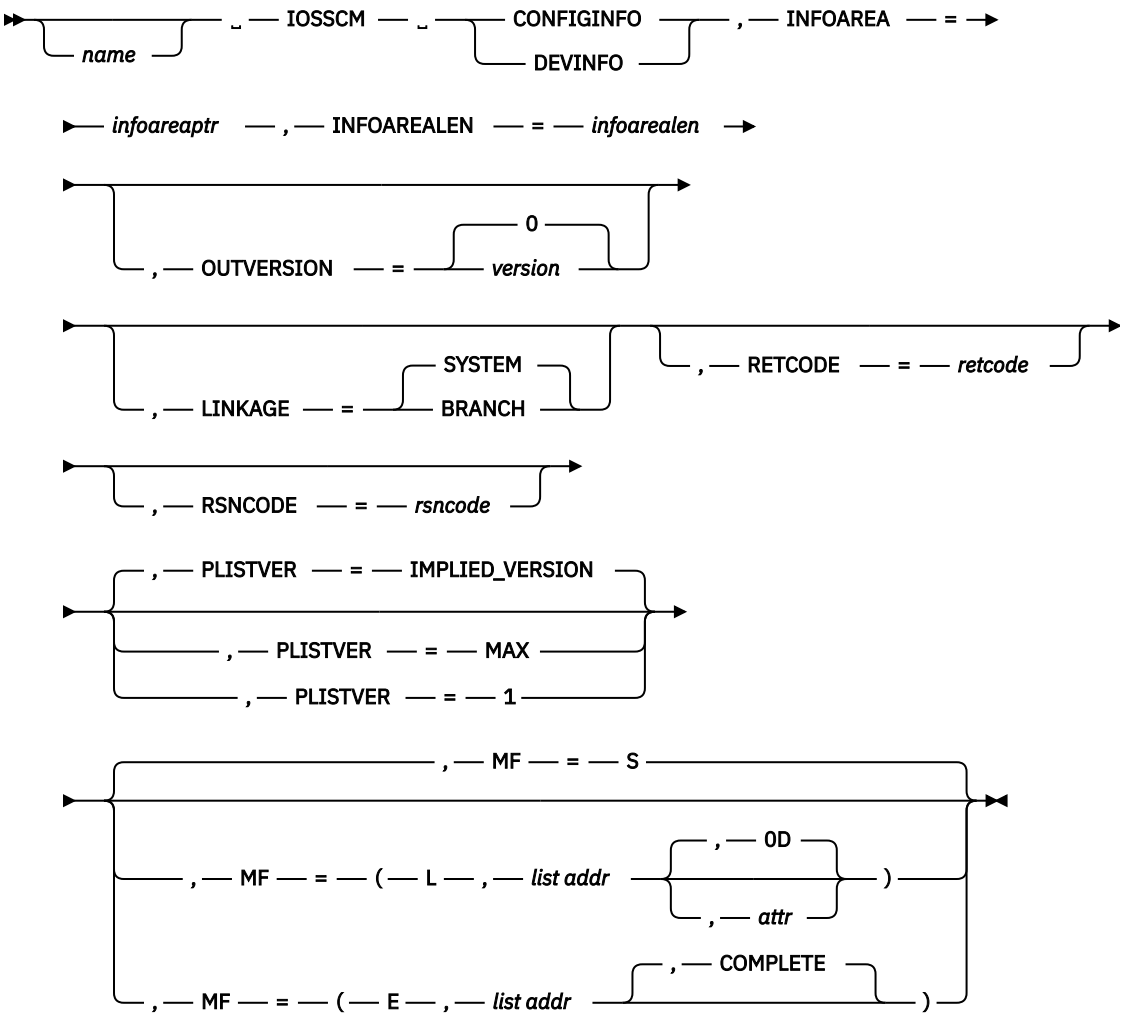
14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSSCM macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

CONFIGINFO
DEVINFO
Specifies the type of SCM information to be returned.

CONFIGINFO
Requests that SCM configuration information be returned. SCM configuration information includes the number of SCM resource parts and the size of an SCM measurement block.

DEVINFO
Requests that SCM device (subchannel) information be returned. SCM device information includes I/O statistics for requests that were issued to each SCM device. Note that the SCM device or subchannel is not associated with a specific Flash Express feature pair; any SCM device can be used to perform I/O to any Flash Express feature pair.

,INFOAREA=*infoareaptr*
On a **CONFIGINFO** request, *infoareaptr* specifies the name (RS-type), or address in register (2) - (12), of a required 8-byte input that contains the address of an area which is to receive the configuration

information. The area must be addressable in the primary address space. The returned **INFOAREA** data is mapped by the IOSDSCCI macro.

On a **DEVINFO** request, *infoareaptr* specifies the name (RS-type), or address in register (2) - (12), of a required 8-byte input that contains the address of an area which is to receive the device information. The area must be addressable in the primary address space. The returned **INFOAREA** data is mapped by the IOSDSCDI macro.

,INFOAREALEN=*infoarealen*
On a **CONFIGINFO** request, *infoarealen* specifies the name (RS-type), or address in register (2) - (12), of a required 4-byte input/output area that contains the length of the configuration information area. If the information area is not large enough to contain all of the data for the requested output version, IOSSCM returns a program error and this field will be updated with the required length.

On a **DEVINFO** request, *infoarealen* specifies the name (RS-type), or address in register (2) - (12), of a required 4-byte input/output area that contains the length of the device information area. The area must be large enough to contain the returned information for each SCM device or subchannel. You can issue a `IOSSCM CONFIGINFO` request to obtain the number of SCM devices or subchannels. If the information area is not large enough to contain all of the data for the requested output version, IOSSCM returns a program error and this field will be updated with the required length.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Optional keyword input that indicates how the service is to be invoked.

SYSTEM
Indicates that a program call (PC) is to be issued. This is the default.

BRANCH
Indicates that a branch-entry linkage is to be generated. See "Environment" on page 1113 for the restrictions on the branch-entry invocation.

Default: SYSTEM

,OUTVERSION=0
,OUTVERSION=*version*
The name (RS-type), or address in register (2) - (12), of an optional 1-byte input that specifies the output version of the output information to be returned. The output version controls the size and format of the returned information. If you specify an output version that is higher than the highest supported version, the highest supported version is used. The INFOAREA output (mapped by the IOSDSCCI or IOSDSCDI macro) contains the version of the returned information.

Note: Currently, version 0 is the only supported value.

Default: 0

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. **PLISTVER** determines which parameter list the system generates. **PLISTVER** is an optional input parameter on all forms of the macro, including the list form. When using **PLISTVER**, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the **PLISTVER** parameter, IMPLIED_VERSION is the default.

MAX
Specifies that you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If your program can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

1
Supports all parameters except those specifically referenced in higher versions.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the **PLISTVER** parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IOSSCM macro returns control to your program:

- GPR 15 (and *retcode*, when you code **RETCODE**) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code **RSNCODE**) contains a reason code.

Table 102 on page 1118 identifies the hexadecimal return and reason codes.

Table 102. Return and reason codes for the IOSSCM macro

Return code	Reason code	Meaning and action
00	–	**Meaning**: The IOSSCM macro completed successfully. **Action**: None.
04	01	**Meaning**: Warning. The requested output version (**OUTVERSION**) is not supported. The information area was updated based on the highest supported output version. **Action**: Examine the output version field in the information area to determine the format of the data that was returned.
08	01	**Meaning**: Program error. An error occurred when the system attempted to reference the area specified by the **INFOAREA** parameter. **Action**: Correct the address specified on the **INFOAREA** parameter and reissue the macro.
08	02	**Meaning**: Program error. The system could not access the caller's parameter list. **Action**: Check to see if your program inadvertently overlaid the parameter list generated by the macro.
08	03	**Meaning**: Program error. The output area specified by the **INFOAREA** parameter is not large enough to hold the requested information. The **INFOAREALEN** field has been updated with the required storage length. **Action**: Obtain storage for the output information area based on the returned **INFOAREALEN** value and reissue the macro.
0C	01	**Meaning**: Environmental error. Storage class memory is not supported on this CPC. **Action**: None.
0C	02	**Meaning**: Environmental error. The IOSSCM service is not available. **Action**: None.
0C	03	**Meaning**: Environmental error. IOSSCM detected that the caller is not in Primary ASC mode. **Action**: None.

Table 102. Return and reason codes for the IOSSCM macro (continued)

Return code	Reason code	Meaning and action
0C	04	**Meaning**: Environmental error. IOSSCM detected that the caller is not enabled for I/O interrupts. **Action**: None.
20	–	**Meaning**: System error. An unexpected error occurred. **Action**: Contact IBM Support.

Chapter 115. IOSSPOF — Check for single points of failure

Description

The IOSSPOF macro is used to check for I/O configuration redundancy of DASD devices or pairs of DASD devices. IOSSPOF verifies redundant hardware components, such that given failure of a hardware component, the availability of the device would be unaffected.

Environment

The requirements for the caller of IOSSPOF are:

Environmental factor	Requirement
Dispatchable unit mode:	Task mode.
Minimum authorization:	Problem state. Any PSW key.
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31 bit or 64 bit If in AMODE 64, specify SYSSTATE AMODE64=YES before starting this macro.
ASC mode:	Primary
Interrupt status:	The caller must be enabled for I/O and external interrupts.
Locks:	The caller cannot hold any locks.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the IOSSPOF macro, the caller does not have to place any information into any general-purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or by using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if the return code is not 0. Otherwise, used as a work register by the system.

1
Used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
Return code.

.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

Some callers depend on register contents to remain the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IOSSPOF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSSPOF.
IOSSPOF	
ƀ	One or more blanks must precede IOSSPOF.
[,xlabel]	An optional symbol, starting in column 1 that is the name on the IOSSPOF macro invocation. The name must conform to the rules for an ordinary assembly language symbol. DEFAULT: No name.
PERFORM_CHECK	
,DEVN1=*xdevn1*	*xdevn1*: RS-type address or register (2) - (12).

Syntax	Description
[,SCHSET1=*xschset1* | 0]	*xschset1*: RS-type address or register (2) - (12). **Default**: 0
[,DSN1=*xdsn1* | *]	*xdsn1*: RS-type address or register (2) - (12). **Default**: *
[,DEVN2=*xdevn2* | *]	*xdevn2*: RS-type address or register (2) - (12). **Default**: *
[,SCHSET2=*xschset2* | 0]	*xschset2*: RS-type address or register (2) - (12). **Default**: 0
[,DSN2=*xdsn2* | *]	*xdsn2*: RS-type address or register (2) - (12). **Default**: *
,VOLSER1=*xvolser1*	*xvolser1*: RS-type address or register (2) - (12).
[,DSN1=*xdsn1* | *]	*xdsn1*: RS-type address or register (2) - (12). **Default**: *
[,VOLSER2=*xvolser2* | *]	*xvolser2*: RS-type address or register (2) - (12). **Default**: *
[,DSN2=*xdsn2* | *]	*xdsn2*: RS-type address or register (2) - (12). **Default**: *
,DEVLIST=*xdevlist*	*xdevlist*: RS-type address or register (2) - (12).
,DEVCOUNT=*xdevcount*	*xdevcount*: RS-type address or register (2) - (12).
[,DSNLIST=*xdsnlist* | *]	*xdsnlist*: RS-type address or register (2) - (12). **Default**: *
[,SCHSETSPEC=NO | YES]	**Default**: NO
,VOLLIST=*xvollist*	*xvollist*: RS-type address or register (2) - (12).
,VOLCOUNT=*xvolcount*	*xvolcount*: RS-type address or register (2) - (12).
[,DSNLIST=*xdsnlist*| *]	*xdsnlist*: RS-type address or register (2) - (12). **Default**: *
[,SPOFAREA=*xspofarea*]	*xspofarea*: RS-type address or register (2) - (12).
[,HCMSG=NO]	
[,HCMSG=YES]	**Default**: NO
[,HANDLE=*xhandle*| *]	*xvolser2*: RS-type address or register (2) - (12). **Default**: *
[,REXX=NO | YES]	**Default**: NO
[,WTO=NO | YES]	**Default**: NO
[,IND_CHECKS=YES | NO | ONLY]	**Default**: YES
[,SWITCH_CHECKS=YES | NO]	**Default**: YES
[,CU_CHECKS=YES | NO]	**Default**: YES
[,RETCODE=*retcode*]	*retcode*: RS-type address or register (2) - (12).
[,RSNCODE=*rsncode*]	*rsncode*: RS-type address or register (2) - (12).
[,PLISTVER=*plistver* MAX]	**Default**: IMPLIED_VERSION

Syntax	Description
[,PLISTVER=*plistver* IMPLIED_VERSION]	
[,MF=S]	**Default**: MF=S
[,MF=(L, *xmfctrl*, *xmfattr*, 0D)]	
[,MF=(M,*xmfctrl*, COMPLETE | NOCHECK)]	**Default**: COMPLETE
[,MF=(E,*xmfctrl*, COMPLETE | NOCHECK)]	**Default**: COMPLETE

Parameters

The parameters are explained as follows:

,PERFORM_CHECK
Perform single point of failure checks. The following parameters are a set of mutually exclusive keys. This set is required; only one key must be specified.

,DEVN1=*xdevn1*
Belongs to a set of mutually exclusive keys. It is the name (RS-type), or address in register (2) - (12), of a halfword input which contains the device number of a device to check for single points of failure.

,SCHSET1=*xschset1*
This parameter is the name (RS-type), or address in register (2) - (12), of an optional byte input that contains the subchannel set of the device associated with the device number in DEVN1.

Default: 0 (Subchannel set zero).

,DSN1=*xdsn1*
This paramter is the name (RS-type), or address in register (2)-(12), of an optional 44-character input that contains a data set name or a description of the data set associated with device specified in DEVN1. The keyword is only used for message generation.

Default: * No dataset will be displayed in any outputted messages.

,DEVN2=*xdevn2*
This is the name (RS-type), or address in register (2) - (12), of an optional halfword input that contains a device number of a device which is used to verify hardware isolation between the devices specified with DEVN1 and this device.

Default: * Pair checking will not be done.

,SCHSET2=*xschset2*
This is the name (RS-type), or address in register (2) - (12), of an optional byte input that contains the subchannel set of the device associated with the device number in DEVN2.

Default: 0 The subchannel is set to zero.

,DSN2=*xdsn2*
This the name (RS-type), or address in register (2) - (12), of an optional 44-character input that contains a data set name or a description of the data set associated with device specified in DEVN2. The keyword is only used for message generation.

Default: * No data set is displayed in the output messages.

,VOLSER1=*xvolser1*
Belongs to a set of mutually exclusive keys. It is the name (RS-type), or address in register (2) - (12), of a 6-character input that contains the VOLSER of the device to check for a single point of failure.

,DSN1=*xdsn1*
This is the name (RS-type), or address in register (2) - (12), of an optional 44-character input that contains a data set name associated with volume specified in VOLSER1. This keyword is only used for message generation.

Default: *

,VOLSER2=*xvolser2*
This is the name (RS-type), or address in register (2) - (12), of a 6-character input that contains a VOLSER of a volume which is used to verify hardware isolation between the volumes specified with VOLSER1 and this volume.

Default: *

,DSN2=*xdsn2*
This is the name (RS-type), or address in register (2) - (12), of an optional 44-character input that contains a data set name associated with volume specified in VOLSER2. This keyword is used for only message generation.

Default: *

,DEVLIST=*xdevlist*
This is the name (RS-type), or address in register (2) - (12), of a 1-byte input that contains the address to an array of fullwords with byte 1 containing zero, byte 2 containing the subchannel set of the device and bytes containing the subchannel set of the device and bytes and 3 and 4 containing the device number of the device to be checked. For example, 0001DE61 represents a device in subchannel set one with a device number of DE61.

Note: Only individual device checks are performed when DEVLIST is specified.

,DEVCOUNT=*xdevcount*
This is the name (RS-type), or address in register (2) - (12), of a fullword input that contains the number of devices in the DEVLIST array.

End of group of keys.

,DSNLIST=*xdsnlist*
This is the name (RS-type), or address in register (2) - (12), of an optional 4-byte input that contains the address of an array of CL44 elements that contain the data set names of the devices that correspond to the DEVLIST parameter. This keyword is used for message generation only.

Default: *

,SCHSETSPEC=NO|YES
This is an optional keyword input that specifies whether the subchannel set value in each entry in the device list has been specified.

Default: NO.

,SCHSETSPEC=NO
This indicates that a subchannel set has not been specified for each device in the device list. When a subchannel set is not specified, the IOSSPOF service uses the device number in bytes 3 and 4 to obtain information about the active device. The IOSSPOF request is rejected if a nonzero subchannel set value appears in any of the device list entries.

,SCHSETSPEC=YES
This indicates that a subchannel set has been specified for each device in the device list. When a subchannel set is specified, the IOSSPOF service uses the subchannel set in byte 2 and the device number in bytes 3 and 4 to obtain information about the device.

Default: NO.

,SPOFAREA=*xspofarea*
This is the name (RS-type), or address in register (2) - (12), of an optional 4-byte output that contains the address that contains the data requested. The data is mapped by IOSDSPOF, and is only valid if the service ended with a 4 or 8 return code. The SPOFAREA is obtained by the service and must be

released by the caller by using the length and subpool specified in the SPOFAREA. The SPOFAREA can be returned in a subpool that is not associated with the issuing task and thus, the caller must not assume the storage is automatically released when the task ends. If the caller is in a PSW key other than key 0-7 when the IOSSPOF service is invoked, the caller must ensure the SPOFAREA storage is accessed while in a PSW key equal to the key of the calling task.

,HCMSG=NO|YES
This is an optional keyword input that specifies whether health checker messages should be issued automatically with this service. HCMSG=YES without a HANDLE is only valid when running under IBM Health Checker for z/OS.

Default: NO.

,HCMSG=NO
Indicates that health checker messages should not be issued.

,HCMSG=YES
Indicates that health checker messages should be issued through HZSFMSG. HCMSG is only valid when the IOSSPOF service is called from a Health Check running under control of the IBM Health Checker for z/OS.

Default: NO.

,HANDLE=*xhandle*
This is the name (RS-type), or address in register (2) - (12), of an optional 16-character input that specifies a handle (token) that identifies the check. This handle is returned through the HANDLE parameter of the HZSADDCK macro for a REMOTE=YES check. HANDLE is required when the service is called from a remote check and is ignored when the service is called from a local check. If IBM Health Checker for z/OS is not running at the time of invocation, then a return code of X'10' with a reason code of '02' will be returned.

Default: * Health checker messages will be issued as a REMOTE=NO call.

,REXX=NO|YES
This is an optional keyword input that specifies whether this is a REXX check.

Default: NO.

,REXX=NO
This indicates that this is not a REXX check.

,REXX=YES
This indicates that this is a REXX check.

Default: NO.

,WTO=NO|YES
This is an optional keyword input that specifies whether WTOs of IOSPFxxxI messages will be issued for this service.

Default: NO.

,WTO=NO
Indicates that WTOs will not be issued for this service.

,WTO=YES
Indicates that WTOs will be issued for this service. The IOSPFxxxI messages will be issued with a ROUTCDE=11.

Default: NO.

,IND_CHECKS=YES|NO|ONLY
This is an optional keyword input that specifies whether single points of failure for individual devices should be checked. For example, checks that are not comparing two devices for mutual single points of failure should be done. This keyword is ignored if a single device is specified. The specific device checks like the following are performed if YES is specified.

- Check to see if a device has only one path available.
- Check to see if the paths of the device share internal hardware subchannel components.

Default: YES.

,IND_CHECKS=YES
Indicates that individual device checks should be made. That is, all checks should be made.

,IND_CHECKS=NO
Indicates that individual device checks should not be made or only pair checks should be made.

,IND_CHECKS=ONLY
Indicates that only individual device checks should be made or no pair checks should be made.

,SWITCH_CHECKS=YES|NO
This is an optional keyword input that specifies whether to check for switch related single points of failure. It applies to individual and pairs checks. The following specific device checks are performed if YES is specified:

- Check if all online CHPIDs are connected to the same switch.
- Check if all devices are connected to the same switch.

Default: YES.

,SWITCH_CHECKS=YES
Indicates that switch related checks should be made.

,SWITCH_CHECK=NO
Indicates that switch related checks should not be made.

,CU_CHECKS=YES|NO
This is an optional keyword input specifies whether to check for control unit-related single points of failure. It applies to individual and pair checks. The following specific device checks are performed if YES is specified:

- Check if all devices are in the same DASD logical subsystem (LSS).
- Check if all devices are in the same physical control unit.
- Check if all devices are sharing the same set of control unit interfaces.

This keyword is ignored if a single device is specified. That is, DEVN1 is specified without DEVN2 or VOLSER1 is specified without VOLSER2.

Default: YES.

,CU_CHECKS=YES
Indicates that control unit-related checks should be made.

,CU_CHECKS=NO
Indicates that control unit-related checks should not be made.

,RETCODE=*retcode*
The name (RS-type) of an optional fullword output variable, or register (2) - (12) or (15), into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value is left in GPR15.

,RSNCODE=*xrsncode*
The name (RS-type) of an optional fullword output variable into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR00, REG0, REG00, or R0 (within or without parentheses), the value is left in GPR 0.

,PLISTVER=*plistver* | MAX | IMPLIED_VERSION
Is an optional byte input decimal value in the "1–1" range that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated.

Note: MAX may be specified instead of a number and will cause the parameter list to be of the largest size currently supported. This size may grow from release to release which possibly may affect the amount of storage needed by your program. If your program can tolerate this, IBM recommends that

you always specify MAX when creating the list form parameter list as this will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

Default: IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version, which allows all of the parameters specified on the invocation to be processed.

,MF=S|L|M|E
An optional keyword input that specifies the macro form.

Default: S.

,MF=S
Specify the standard form of the macro. The "S" form builds an inline parameter list and generates the macro invocation to transfer control to the service. Full checking for required macro keys is done along with supplying defaults for omitted optional parameters.

,MF=(L,*xmfctrl*,*xmfattr*, 0D)
Specifies the list form of the macro. The "L" form defines a storage area for the parameter list. Only the PLISTVER key can be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

xmfctrl
A required input. It is the name of a storage area for the parameter list.

xmfattr
An optional 60-character input string that varies with 1 - 60 characters. Use it to force boundary alignment of the parameter list. Use only 0F or 0D. The default is 0D, which forces the parameter list to a doubleword boundary.

,MF=(M,*xmfctrl*,COMPLETE | NOCHECK)
Specifies the modify form of the macro. The "M" form generates code to put the parameters into the parameter list specified by *xmfctrl*.

xmfctrl
A required input. It is the name (RS-type), or address in register (1) - (12), of a storage area for the parameter list.

,COMPLETE | NOCHECK
An optional keyword input that specifies the degree of macro parameter syntax checking.

Default: COMPLETE.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Checking for required macro keys is not done or defaults are not supplied for omitted optional parameters.

,MF=(E,*xmfctrl*,COMPLETE | NOCHECK
Specifies the execute form of the macro. The "E" form generates code to put the parameters into the parameter list specified by *xmfctrl* and invoke the wanted service.

xmfctrl
A required input. It is the name (RS-type), or address in register (1) - (12), of a storage area for the parameter list.

,[COMPLETE | NOCHECK]
An optional keyword input that specifies the degree of macro parameter syntax checking.

Default: COMPLETE.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Checking for required macro keys is not done or defaults are not supplied for omitted optional parameters.

ABEND codes

None.

Return codes

Return codes, in hexadecimal, from the IOSSPOF macros are as follows:

Hexadecimal Return Code	Equate Symbol Meaning and Action
00	**Equate Symbol**: SPOF_RC_Ok **Meaning**: No single points of failure detected. **Action**: None.
04	**Equate Symbol**: SPOF_RC_SomeChecksFailed **Meaning**: The service might not perform all checks specified, but no single points of failure were detected. **Action**: Some checks can fail due to switch devices not being online at the time of the check. All switch devices must be online to determine if control unit interfaces are single point of failure free.
08	**Equate Symbol**: SPOF_RC_SPOFFound **Meaning**: Single points of failure were detected. **Action**: Refer to IOSPFxxxI message for action. **Hex Reason Code** **Meaning/Action** **00** **Equate Symbol**: SPOF_RSN_AllDevicesFound **Meaning**: While a single point of failure was discovered all devices were found. **01** **Equate Symbol**: SPOF_RSN_DeviceNotFound **Meaning**: Single points of failure were detected, and one or more of the devices specified are not found.

Hexadecimal Return Code	Equate Symbol Meaning and Action
0C	**Equate Symbol**: SPOF_RC_ProgramError **Meaning**: Program error. **Action**: None. **Hex Reason Code** **Meaning/Action** **01** **Equate Symbol**: SPOF_RSN_InvalidParmListVers **Meaning**: It was discovered that the macro was invoked with an invalid parameter list. **Action**: Specify a valid parameter list version. **02** **Equate Symbol**: SPOF_RSN_InvalidCount **Meaning**: The number of devices specified by way of the DEVCOUNT parameter or volume serial numbers by way of the VOLCOUNT parameter is invalid. **Action**: Change DEVCOUNT or VOLCOUNT to be less than 65536 and greater than zero. **03** **Equate Symbol**: SPOF_RSN_ImproperModes **Meaning**: IOSSPOF was invoked in an improper mode. **Action**: See environment specification for what modes IOSSPOF can be invoked and invoke only in supported modes. **04** **Equate Symbol**: SPOF_RSN_ImproperDevlistEntry **Meaning**: A device in the device list did not match the format '000sdddd' where '000s' is the subchannel set and 'dddd' is the device number. **Action**: Adjust DEVLIST parameter to match the format. **05** **Equate Symbol**: SPOF_RSN_BadParmListAccess **Meaning**: Abend accessing parameter list. **Action**: Verify that the parameters can be accessed by invokers key.
10	**Equate Symbol**: SPOF_RC_EnvironError **Meaning**: Environmental error. **Action**: None. **Hex Reason Code** **Meaning/Action** **01** **Equate Symbol**: SPOF_RSN_IOSSPOFNotAvail **Meaning**: The IOSSPOF service is not available currently. **Action**: Wait until the IOSAS address space is available. **02** **Equate Symbol**: SPOF_RSN_HlthChkerNotAvail **Meaning**: The Health Checker environment isn't available and is available if HCMSG=YES and HANDLE is specified. **Action**: Start the Health Checker if HCMSG=YES is required.
20	**Equate Symbol**: SPOF_RC_SystemError **Meaning**: System error. **Action**: None.

Return and reason codes

Macro IOSDSPOF provides equate symbols for the return and reason codes.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00	00	Always set.
04	00	Always set.
08	00	Single points of failure were detected. All devices are found.
08	01	Single points of failure were detected, and one or more of the devices specified are not found.
0C	01	Incorrect parameter list version.
0C	02	The number of devices specified through the DEVCOUNT parameter or volume serial numbers specified through the VOLCOUNT parameter is incorrect.
0C	03	The caller is in an improper mode when invoked.
0C	04	A device in the device list does not match the format '000sdddd' where '000s' is subchannel set and 'dddd' is the device number.
0C	05	Abend accessing parameter list.
10	01	The IOSSPOF service is not available currently.
10	02	The Health Checker service is not available currently.
20	00	Always set.

IOSSPOF - List form

Use the list form of the IOSSPOF macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

Syntax

The list form of the IOSSPOF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSSPOF.
IOSSPOF	
ƀ	One or more blanks must follow IOSSPOF.
[,PLISTVER={*xplistver* | MAX | IMPLIED_VERSION}]	**Default**: IMPLIED_VERSION
,MF=(L,*xmfctrl*,*xmfattr*, 0D),	*xmfctrl*: RS-type name or address in register (2)–(12). *xmfattr*: Any text up to 60 characters. **Default**: 0D

Parameters

The parameters are explained under the standard form of the IOSSPOF macro, with the following exception:

,MF=(L,*xmfctrl*{,*xmfattr*|0D})

L specifies the list form of the macro. The "L" form defines an area to be used for the parameter list. Only the PLISTVER key may be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

,xmfctrl

This required input is the RS-type name, or address in register (1)–(12), of a storage area for the parameter list.

***,xmfattr*|0D**

This is an optional 60 character input string which is used to force boundary alignment of the parameter list. Use only 0F or 0D.

The default is 0D, which forces the parameter list to a doubleword boundary.

IOSSPOF - Execute form

Use the execute form of the IOSSPOF macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSSPOF macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSSPOF.
IOSSPOF	
ƀ	One or more blanks must follow IOSSPOF.
[,PLISTVER={*xplistver* | MAX | IMPLIED_VERSION}]	**Default**: IMPLIED_VERSION
,MF=(L,*xmfctrl*,*xmfattr*, 0D),	*xmfctrl*: RS-type name or address in register (2)–(12). *xmfattr*: Any text up to 60 characters. **Default**: 0D

Parameters

The parameters are explained under the standard form of the IOSSPOF macro, with the following exception:

,MF=(E,*xmfctrl*{,COMPLETE})

E specifies the execute form of the macro. The "E" form generates code to put the parameters into the parameter list specified by xmfctrl and provides full syntax checking with default setting.

,xmfctrl

This required input is the RS-type name, or address in register (1) – (12), of a storage area for the parameter list.

,COMPLETE

An optional keyword which specifies the degree of macro parameter syntax checking. When complete checking is enabled, required parameters are checked and defaults are supplied for omitted optional parameters.

Default: COMPLETE

Chapter 116. IOSUPFA — Obtain address of the UCB prefix extension segment

Note: The UCBLOOK macro is the preferred programming interface.

Description

The IOSUPFA macro obtains the address of the UCB prefix extension segment. To map the UCB prefix extension segment, use the IOSDUPFX mapping macro.

The IOSUPFA macro provides faster performance than the UCBLOOK macro; however, if the caller uses UCBLOOK to obtain several addresses in the same invocation, UCBLOOK might provide better performance than an IOSUPFA macro and an IOSCMXA macro. The UCBLOOK macro also validates input parameters and provides recovery.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	The input parameter must be in the primary address space. If the caller is disabled, the parameter list must reside in fixed or disabled reference (DREF) storage.

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be deleted. (If the caller issues a UCBLOOK macro with the PIN parameter to pin the UCB, use the UCBLOOK UCBPXPTR parameter rather than the IOSUPFA macro.)

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

None.

Input register information

Before issuing the IOSUPFA macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14
Return address

15
Used as a work register by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

Performance implications

None.

Syntax

The standard form of the IOSUPFA macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSUPFA.
IOSUPFA	
ƀ	One or more blanks must follow IOSUPFA.
UCBPTR=*ucbptr addr*	*ucbptr addr*: RX-type address or register (2) - (12).
,UCBPADDR=*ucbpaddr addr*	*ucbpaddr addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

UCBPTR=*ucbptr addr*
Specifies the address of a fullword field that contains the address of the UCB common segment. This address must not be associated with a copy of the UCB.

,UCBPADDR=*ucbpaddr addr*
Specifies the address of a fullword field in which the system returns the address of the UCB prefix extension segment. Use the IOSDUPFX mapping macro to map the UCB prefix extension segment.

ABEND codes

None.

Return and reason codes

None.

IOSUPFA - List form

Use the list form of the IOSUPFA macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros.

Syntax

The list form of the IOSUPFA macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSUPFA.
IOSUPFA	
ƀ	One or more blanks must follow IOSUPFA.
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default:** 0D

Parameters

The parameters are explained under the standard form of the IOSUPFA macro with the following exception:

MF=(L,*list addr*)
MF=(L,*list addr*,*attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOSUPFA macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

IOSUPFA - Execute form

Use the execute form of the IOSUPFA macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the IOSUPFA macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSUPFA.
IOSUPFA	
ƀ	One or more blanks must follow IOSUPFA.
UCBPTR=*ucbptr addr*	*ucbptr addr*: RX-type address or register (2) - (12).
,UCBPADDR=*ucbpaddr addr*	*ucbpaddr addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained under the standard form of the IOSUPFA macro with the following exception:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the IOSUPFA macro.

list addr specifies the area that the system uses to contain the parameters.

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

Chapter 117. IOSUPFR — Obtain address of the UCB prefix extension segment

Description

Use the IOSUPFR macro to obtain the address of the UCB prefix extension segment. To map the UCB prefix extension segment, use the IOSDUPFX mapping macro.

UCBLOOK and IOSUPFA macros also provide this function. However, IOSUPFR provides an alternative for passing parameters (that is, in general purpose register (GPR) 1 rather than in a parameter list). For guidance about obtaining UCB information, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	None

Programming requirements

The caller must pass a valid captured or actual UCB address.

The caller must pin the UCB or otherwise guarantee that the UCB will not be dynamically deleted.

The caller must supply recovery to handle any unexpected errors, such as abends.

Restrictions

None.

Input register information

Before issuing the IOSUPFR macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

1
Address of UCB common segment

Before issuing the IOSUPFR macro, the caller does not have to place any information into any access register (AR).

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the UCB prefix extension

2-13
Unchanged

14-15
Used as work registers by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IOSUPFR macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSUPFR.
IOSUPFR	
ƀ	One or more blanks must follow IOSUPFR.
MF=(S)	**Default**: S

Parameters

The parameters are explained as follows:

MF=(S)
Specifies the standard form of the macro. This parameter is optional.

ABEND codes

None.

Return and reason codes

None.

Chapter 118. IOSVRYSW — Vary switch service

Description

IOSVRYSW provides an interface to the VARY SWITCH process to configure a switch port online or offline to Dynamic Channel Path Management (DCM). Invoking this interface for a switch port also causes the specific managed device paths to be varied online or offline. An online request causes the managed channel paths to become eligible to DCM. An offline request causes the managed channel paths to be removed from the control units connected to the managed channel path IDs (CHPIDs) at the specified ports. This command affects only managed device paths. Non-managed paths must be varied online or offline separately.

Note: VARY SWITCH command performs the same function when it is issued from a console. For more information, see Placing a switch port online or offline in *z/OS MVS System Commands*.

Macro IOSDVSAP maps each element of the array of resource elements that is passed to the VARY SWITCH programming interface. Each element is created by a separate IOSVRYSW BUILD invocation and represents a vary switch port online, offline, or offline and unconditional request.

Note: The caller is responsible for obtaining the right amount of storage for the array of resource elements before the first IOSVRYSW BUILD request.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, or PSW key 0-7, or APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

Programs invoking this interface must include mapping macro IOSDVSAP.

Restrictions

None.

Input register information

Before issuing the IOSVRYSW macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Register
Contents

0
Undefined

1
Used by the service

2-13
Undefined

14-15
Used by the service

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code (valid for TYPE=INVOKE, unpredictable otherwise)

1
Unpredictable

2-13
Unchanged

14
Unpredictable

15
Return code (valid for TYPE=INVOKE, unpredictable otherwise)

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Performance implications

None.

Syntax

The standard form of the IOSVRYSW macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSVRYSW.
IOSVRYSW	
ƀ	One or more blanks must follow IOSVRYSW.

Syntax	Description
TYPE=BUILD	
,REQUEST=ONLINE	**Default:** ONLINE
,REQUEST=OFFLINE	
,OPTION=UNCOND	**Default:** NONE
,SWITCHDEV=*switchdev*	*switchdev*: RS-type name, or address in register (2)-(12).
,PORTADDR=*portaddr*	*portaddr*: RS-type name, or address in register (2)-(12).
INVOKE	
,DATAADDR=*dataaddr*	*dataaddr*: RS-type name, or address in register (2)-(12).
,DATANUM=*datanum*	*datanum*: RS-type name, or address in register (2)-(12).
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12). Can only be specified with an INVOKE request.
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12). Can only be specified with an INVOKE request.
,MF=(E,*list addr*)	*list addr*: RS-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained as follows:

TYPE=BUILD
Specifies a required keyword input which indicates that the macro is being invoked to construct vary switch parameters.

REQUEST=ONLINE
REQUEST=OFFLINE
Specifies an optional keyword input which indicates the type of request to process.

- **ONLINE**: The request is to configure a switch port online to Dynamic Channel Path Management. The default is **ONLINE**.

- **OFFLINE**: The request is to configure a switch port offline to Dynamic Channel Path Management.

OPTION=UNCOND
Specifies an optional keyword input which indicates an additional option to be processed along with an OFFLINE request.

UNCOND: This option is used to specify an UNCOND request on the VARY PATH commands invoked as a result of the VARY SWITCH request. Adding the UNCOND keyword to a VARY PATH,OFFLINE command results in the system taking offline the last path to devices that are online but unallocated.

SWITCHDEV=*switchdev*
Specifies an RS-type name, or address in register (2)-(12), of a required halfword input which indicates the switch device number to be affected.

PORTADDR=*portaddr*
Specifies an RS-type name, or address in register (2)-(12), of a required byte input which indicates the port address to be affected.

TYPE=INVOKE
Specifies a required keyword input which indicates to perform the requested Vary Switch function built by one or multiple IOSVRYSW BUILD requests.

DATAADDR=*dataaddr*
Specifies an RS-type name, or address in register (2)-(12), of a required 4 byte input that contains the address to the array of resource elements to be processed. Each element is created by an IOSVRYSW BUILD invocation and is mapped by mapping macro IOSDVSAP.

DATANUM=*datanum*
Specifies an RS-type name, or address in register (2)-(12), of a required fullword input that contains the number of elements in the array of resource elements to be processed.

RETCODE=*retcode*
Specifies an RS-type name of an optional fullword output variable, or register (2)-(12), into which the return code is to be copied from GPR 15.

Note: This keyword can only be specified with an INVOKE request.

RSNCODE=*rsncode*
Specifies an RS-type name of an optional fullword output variable, or register (2)-(12), into which the reason code is to be copied from GPR 0.

Note: This keyword can only be specified with an INVOKE request.

MF=(E,*list addr*)
MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the IOSVRYSW macro.

list addr
Specifies the area that the system uses to contain the parameters.

COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters. **COMPLETE** is the default.

ABEND codes

None.

Return and reason codes

Note: There are no return or reason codes for TYPE=BUILD.

When the system returns control to the caller, GPR 15 (and *retcode*, when you code RETCODE) contains a return code.

The following table identifies the hexadecimal return codes:

Table 103. Return Codes for the IOSVRYSW Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning:** Successful completion. **Action:** None required.
10	**Meaning:** An unexpected error occurred in vary switch processing. **Action:** Verify the configuration in effect and resubmit the request. If the request fails again for the same reason, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
FF04	**Meaning:** Storage passed on the DATAADDR was not accessible by the service. **Action:** Verify that accessible storage is being passed.
FF08	**Meaning:** The attempt to queue a work element to the IOS address space failed. Request is currently not able to be performed. **Action:** Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
FF0C	**Meaning:** VSAP data is readable but not valid. **Action:** Verify that the correct data is being passed.
FF10	**Meaning:** Caller is not in a valid environment to invoke the IOSVRYSW API. **Action:** Insure that the caller is running in the correct environment.
FF14	**Meaning:** Module IOSVVSWF suffered a catastrophic error. Function could not be processed. **Action:** Search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
FF18	**Meaning:** Module IOSVVSWF could not establish a recovery environment. **Action:** Resubmit the request. If the request fails again for the same reason, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Hexadecimale Reason Codes: None.

IOSVRYSW—List form

Use the list form of the IOSVRYSW macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

Syntax for IOSVRYSW—List form

The list form of the IOSVRYSW macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSVRYSW.
IOSVRYSW	

Syntax	Description
ƀ	One or more blanks must follow IOSVRYSW.
MF=(L,*list addr*)	*list addr*: Symbol.
MF=(L,*list addr*,*attr*)	*attr*: 1- to 60-character input string
MF=(L,*list addr*,0D)	**Default**: 0D

Parameters for IOSVRYSW—List form

The parameters are explained as follows:

MF=(L,*list addr*)
MF=(L,*list addr*,*attr*)
MF=(L,*list addr*,0D)
Specifies the list form of the IOSVRYSW macro.

list addr is the name of a storage area to contain the parameters.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

Examples

An example of issuing three IOSVRYSW TYPE=BUILD and an IOSVRYSW TYPE=INVOKE invocations to process three switch ports.

Initial Setup:

- Define the list form of the macro
- Obtain storage for the array of resource elements (See mapping macro IOSDVSAP in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary))
- Establish addressability to the area
- Clear the area
- Set up a pointer, pointing to the beginning of the area
- Base VSAP_RESOURCE structure on the pointer that points to the beginning of this area.
- For EACH port to be altered (for each request), issue IOSVRYSW TYPE=BUILD:
  - IOSVRYSW TYPE=BUILD,
    REQUEST=ONLINE,
    SWITCHDEV=*switch*
    PORTADDR=*port_address*
  - Advance the pointer by length of VSAP_RESOURCE to the next slot in the array
  - IOSVRYSW TYPE=BUILD,
    REQUEST=ONLINE,
    SWITCHDEV=*switch*
    PORTADDR=*port_address*
  - Advance the pointer by length of VSAP_RESOURCE to the next slot in the array
  - IOSVRYSW TYPE=BUILD,
    REQUEST=ONLINE,

SWITCHDEV=*switch*
PORTADDR=*port_address*

- Issue IOSVRYSW TYPE=INVOKE to process the requests, passing in the pointer to the array of resource elements and the number of elements to the processing module.
- IOSVRYSW TYPE=INVOKE,
  DATAADDR=pointer to the array of resource elements,
  DATANUM=number of elements to be processed
  (ports to be altered),
  RETCODE=RETURN_CODE,
  RSNCODE=REASON_CODE,
  MF=(E, IOSVRYSW_LIST)

Chapter 119. IOSWITCH — IOS switch information service

Description

IOSWITCH provides a service which callers outside the IOS address space can use to obtain physical topology information about a specific switch and its ports.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or Supervisor state. Any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks may be held.
Control parameters:	Must be in the primary address space.

Programming requirements

None.

Restrictions

The invoker must not hold any locks which would prevent this service from obtaining the IOSYNCH lock. The service must not be invoked until after the IOS space-switching PC table has been established.

Input register information

Before issuing the IOSWITCH macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

The contents of registers 14 through 1 are altered during processing.

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Unpredictable (Used as a work register by the system)

2-13
Unchanged

14
Unpredictable (Used as a work register by the system)

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Unpredictable (Used as work registers by the system)

2-13
Unchanged

14-15
Unpredictable (Used as work registers by the system)

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IOSWITCH macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede IOSWITCH.
IOSWITCH	
␣	One or more blanks must follow IOSWITCH.
SWITCH=*switch*	*switch*: Symbol up to 4 characters long.
,SWITCHAREA=*switcharea*	*switcharea*: RS-type address or address in register (2) - (12).
,SWITCHLEN=*switchlen*	*switchlen*: RS-type address or address in register (2) - (12).
,SUBPOOL=*subpool*	*subpool*: RS-type address or address in register (2) - (12).
,OFFLINE	

Syntax	Description
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	*list addr*: RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSWITCH macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

SWITCH=*switch*
A required 4-character input parameter containing the switch device number.

SWITCHAREA=*switcharea*
A required 4-byte output parameter that will receive the address of the switch data area. The storage for this data must be released by the caller. This data is mapped by IOSDSWTD.

SWITCHLEN=*switchlen*
A required fullword output parameter that will receive the length, in bytes, of the switch data area.

SUBPOOL=*subpool*
The name (RS-type), or address in register (2) - (12), of a required halfword input parameter that identifies the subpool to be used for obtaining storage for the switch data area.

See the list of subpool characteristics in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about authorization requirements for specific subpools.

When the calling program is unauthorized, storage is obtained in the specified subpool, provided that the caller is permitted to use that subpool. Storage will be obtained in the caller's key; however, the resulting key will be set according to the rules for the specified subpool, as documented in *z/OS MVS Programming: Assembler Services Guide*. Valid subpools are: 0 - 127, 131, and 132.

When the calling program is authorized, storage is obtained in key 0. Valid subpools are: 226, 227, 228, 231, 239, 241, 245, 247, and 248.

OFFLINE
An optional keyword that indicates that data will be returned for the switch device even if the device is offline. Note that if the device is in fact offline, the data may be outdated.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When the IOSWITCH macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

Table 104. Return and reason codes for the IOSWITCH macro

Hexadecimal return code	Reason codes, meaning and action
00	IOSWITCH completed successfully.
04	Warning. **Reason code** **Meaning/action** **01** **Meaning:** The switch device provided by the caller is offline. No data is returned. **Action:** To obtain data for the offline switch, use the OFFLINE keyword. **02** **Meaning:** The IOSWITCH service is not enabled at this time. No data is returned. **Action:** Try the service again at a later time. **03** **Meaning:** The switch table is not available. No data is returned. **Action:** Check dynamic channel path management status, as it pertains to the switch table availability.

Table 104. Return and reason codes for the IOSWITCH macro (continued)

Hexadecimal return code	Reason codes, meaning and action
08	Program error. **Reason code** **Meaning/action** **01** **Meaning:** An authorized calling program specified an unauthorized subpool or an unsupported authorized subpool. **Action:** Correct the subpool and reissue the IOSWITCH macro. Authorized programs are restricted to subpools 226, 227, 228, 231, 239, 241, 245, 247, and 248. For a list of subpool characteristics, see *z/OS MVS Programming: Authorized Assembler Services Guide*. **02** **Meaning:** The switch device number provided by the caller is not in the switch table. **Action:** Correct the switch device number and reissue the IOSWITCH macro. **03** **Meaning:** Program error. An error occurred in accessing the caller's parameter list. **Action:** Ensure that the storage area for the parameter list is addressable in the caller's primary address space using the key of the caller. **04** **Meaning:** An unauthorized calling program specified an authorized subpool or an unsupported unauthorized subpool. **Action:** Correct the subpool and reissue the IOSWITCH macro. Unauthorized programs are restricted to subpools 0 - 127, 131, and 132. For a list of subpool characteristics, see *z/OS MVS Programming: Assembler Services Guide*.
20	System error. An unexpected error occurred.

Chapter 120. IOSZHPF — zHPF channel program capabilities service

Description

The IOSZHPF macro provides information about the zHPF capabilities of a device from the operating system, processor, channel, and device point of view. The information returned is mapped by IOSDZHPF and reflects the minimum capability of all of the online channels for a device. For example, if a device has two online channels and one channel supports a new function and the other does not, that function will not be reported.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or supervisor state. Any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	If the caller is disabled, the parameter list must reside in fixed or disabled reference (DREF) storage.

Programming requirements

Users of this macro must make sure that the UCB will not be deleted. The application must pin the UCB, or make sure that the environment it is executing in, will not allow the UCB to be deleted.

This service will not have any recovery. The user must supply recovery to handle any unexpected errors.

Restrictions

None.

Input register information

Before issuing the IOSZHPF macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a 144 byte save area

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

Performance implications

None.

Syntax

The standard form of the IOSZHPF macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSZHPF.
IOSZHPF	
ƀ	One or more blanks must follow IOSZHPF.
INFOAREA=*infoarea*	*infoarea*: RX-type address or register (2) - (12).
,UCBPTR=*ucbptr*	*ucbptr*: RX-type address or register (2) - (12).
,DEVINFO=NO	**Default**: NO
,DEVINFO=YES	
,LINKAGE=BRANCH	
,RETCODE=*xretcode*	*xretcode*: RX-type address or register (2) - (12).
,RSNCODE=*xrsncode*	*xrsncode*: RX-type address or register (2) - (12).

Syntax	Description
,PLISTVER={*xplistver* | MAX | IMPLIED_VERSION}	**Default**: PLISTVER=IMPLIED_VERSION

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IOSZHPF macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

INFOAREA=*infoarea*
The name (RS-type), or address in register (2)-(12), of a required 32 character input into which IOSZHPF is to return the zHPF channel program information. This area is mapped by IOSDZHPF.

,UCBPTR=*ucbptr*
The name (RS-type), or address in register (2)-(12), of a required 4 byte input that contains the address of the UCB (common segment address mapped by IEFUCBOB) whose zHPF information is to be obtained.

,DEVINFO=*NO*
,DEVINFO=*YES*
Specifies whether the device-related zHPF capabilities should be returned.

NO
The device-related zHPF capabilities should not be returned. NO is the default.

YES
The device-related zHPF capabilities should be returned.

,LINKAGE=BRANCH
An optional keyword input that indicates that branch-entry linkage should be issued for the routine invocation.

,RETCODE=*xretcode*
The name (RS-type), or register (2)-(12), of an optional fullword output into which the return code is to be copied from GPR 15.

,RSNCODE=*xrsncode*
The name (RS-type), or register (2)-(12), of an optional fullword output into which the reason code is to be copied from GPR 0.

,PLISTVER=*xplistver* | MAX | IMPLIED_VERSION
An optional byte input decimal value in the "1-1" range that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number and will cause the parameter list to be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as this will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

Default: IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

ABEND codes

None.

Return and reason codes

When the IOSZHPF macro returns control to your program:

- GPR 15 (and *xretcode* when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *xrsncode* when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

Return Code	Meaning
00	Successful completion
04	Warning **Reason Code** **Meaning** **01** zHPF is disabled for the device.
08	Program error **Reason Code** **Meaning** **01** The UCB address provided by the caller does not represent a valid UCB.

Chapter 121. IQPINFO — Obtain PCIe information

Description

The IQPINFO macro provides PCIe-related information, including any performance statistics.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in the primary address space.

Programming requirements

None.

Restrictions

Be aware that, with the IBM z13® enhancement of the IQPINFO service, it is possible that the required buffer length can change depending on the device (such as RoCE or zEDC devices) because different devices each return their specialized data. Therefore, callers of the IQPINFO service must be prepared that the required length that is returned in the PERFDATALEN parameter might not be valid for all devices.

Input register information

Before issuing the ENQ macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller of the IQPINFO macro, the general purpose registers (GPRs) contain:

Register
Contents

0
Unchanged

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller of the IQPINFO macro, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service and restore them after the system returns control.

Performance implications

None.

Syntax

The IQPINFO macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IQPINFO.
IQPINFO	
ƀ	One or more blanks must follow IQPINFO.
PERFDATA	
,FCNINDEX=*xfcnindex*	*xfcnindex*: RS-type address or address in register (2) - (12).
,PERFDATAPTR=*xperfdataptr*	*xperfdataptr*: RS-type address or address in register (2) - (12).
,PERFDATALEN=*xperfdatalen*	*xperfdatalen*: RS-type address or address in register (2) - (12).
,LINKAGE=BRANCH	Default: BRANCH
,RETCODE=*xretcode*	*xretcode*: RS-type address or address in register (2) - (12) or (15).

Syntax	Description
,RSNCODE=*xrsncode*	*xrsncode*: RS-type address.
,PLISTVER=*xplistver*	
,PLISTVER=IMPLIED_VERSION	Default: IMPLIED_VERSION
,MF=S	Default: MF=S
,MF=(L,*xmfctrl,xmfattr* | 0D)	
,MF=(E,*xmfctrl*,COMPLETE)	

Parameters

The parameters are explained as follows:

PERFDATA
Obtains PCIe-related performance information.

,FCNINDEX=*xfcnindex*
Specifies the RS-type address or address in register (2) - (12) of a required fullword input of PCIe function index to start the search from. The IQPINFO service returns performance data for the next function in the PCIe function table after the supplied index value.

If an allocated PCIe function is found in the table after the supplied PCIe function index, the XFCNINDEX field is set to the index of the PCIe function upon which the performance data is returned. If an allocated PCIe function is not found after the supplied index, the XFCNINDEX field is set to 0. To get information for all allocated PCIe functions, this service should be called continuously starting with a 0 function index until a 0 function index is returned.

,PERFDATAPTR=*xperfdataptr*
Specifies the RS-type address or address in register (2) - (12) of a required 8-byte input that contains the address of an area which is to receive the performance data. This area must be addressable in the primary address space. This area is mapped by the IQPYPERF mapping macro.

,PERFDATALEN=*xperfdatalen*
Specifies the RS-type address or address in register (2) - (12) of a required fullword input that contains the length of the area to receive the performance data. If the supplied length is not large enough to contain the data to be returned, the supplied *xperfdatalen* field is set to the length required to fit all of the data to be returned.

,LINKAGE=BRANCH
An optional keyword input that indicates the linkage that should be generated for the routine invocation. The default is LINKAGE=BRANCH.

LINKAGE=BRANCH requests branch-entry invocation.

,RETCODE=*xretcode*
The name (RS-type) of an optional full-word output variable, or register (2)-(12) or (15), into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

,RSNCODE=*xrsncode*
The name (RS-type) of an optional fullword output variable into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

,PLISTVER=*xplistver*
,PLISTVER=IMPLIED_VERSION
An optional byte input decimal value (with a value of 1) that specifies the macro version. PLISTVER is the only key allowed on the list form of MF and determines which parameter list is generated. Note that MAX may be specified instead of a number, and the parameter list will be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

The default is IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

,MF=S
,MF=(L,*xmfctrl*,*xmfattr* | 0D)
,MF=(E,*xmfctrl*,COMPLETE)
An optional keyword input which specifies the macro form. The default is S.

,MF=S
Specifies the standard form of the macro. The 'S' form generates code to put the parameters into an in-line parameter list and invoke the desired service. Full checking for required macro keys is done along with supplying defaults for omitted optional parameters.

,MF=(L,*xmfctrl*,*xmfattr* | 0D)
Specifies the list form of the macro. The 'L' form defines an area to be used for the parameter list. Only the PLISTVER key may be specified on the invocation. All other macro parameters are flagged as errors. If PLISTVER is not specified, the original parameter list definition is used.

,xmfctrl
A required input. It is the name of a storage area for the parameter list.

***,xmfattr* | 0D**
An optional 60-character input string that varies from 1 to 60 characters. Use it to force boundary alignment of the parameter list. Use only 0F or 0D. The default is 0D, which forces the parameter list to a doubleword boundary.

,MF=(E,*xmfctrl*,COMPLETE)
Specifies the execute form of the macro. The 'E' form generates code to put the parameters into the parameter list specified by xmfctrl and provides full syntax checking with default setting.

,xmfctrl
A required input. It is the name (RS-type) or address in register (1) - (12) of a storage area for the parameter list.

,COMPLETE
An optional keyword input which specifies the degree of macro parameter syntax checking. The default is COMPLETE.

Return codes

Table 105. Return codes for the IQPINFO macro

Hexadecimal return code	Meaning and action
00	**Meaning**: The requested function was successfully executed. **Action**: None.
08	**Meaning**: The supplied data area that receives the output from the IQPINFO macro is not large enough to hold all of the data to be returned. The supplied length field is set to contain the size of the area that is required to fit all of the data to be returned. **Action**: Increase the size of the supplied data area to be at least the size that is returned in the PERFDATALEN parameter and retry the IQPINFO request.

Table 105. Return codes for the IQPINFO macro (continued)

Hexadecimal return code	Meaning and action
0C	**Meaning**: Caller is not authorized to use the IQPINFO macro. **Action**: The caller of IQPINFO must have proper authorization (supervisor state and running with PSW key 0).
10	**Meaning**: Unexpected error occurred during IQPINFO processing. **Action**: Retry the IQPINFO request. If the problem persists, record the return codes and supply them to the appropriate IBM support personnel.
14	**Meaning**: The requested IQPINFO PLISTVER number is incorrect. **Action**: Specify the correct PLISTVER number in the invocation of the IQPINFO macro. If PLISTVER(MAX) is specified, the parameter list will be of the largest size currently supported.
18	**Meaning**: The supplied data area address is not on a doubleword boundary. **Action**: Provide a data area with an address that is on a 8 byte boundary (that is, the last 3 bits of the address must be 0).
20	**Meaning**: The IQPINFO service is not available. PCIE services have not been enabled on the current system. **Action**: Report the problem to the system programmer.

Chapter 122. IRDFSD — FICON switch data services

Description

The FICON Switch Data macro service is used to obtain statistical counters from FICON switch devices.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or Supervisor state or any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None

Restrictions

- No locks can be held
- Must not be in an environment that would prevent EXCP from being issued
- Must be authorized

Input register information

Register
Contents

0–15
Undefined

Before issuing the IRDFSD macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
Reason Code

1
Used as a work register by the system.

2-13
Restored

14
Used as a work register by the system.

15
Return code

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IRDFSD macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IRDFSD.
IRDFSD	
ƀ	One or more blanks must follow IRDFSD.
DEVICE=ALL	**Default:** DEVICE=ALL
DEVICE=SINGLE	
,DEVICENUMBER=*devicenumber*	Required with DEVICE=SINGLE
,COUNTERS=DEFAULT	**Default:** COUNTERS=DEFAULT
COUNTERS=ALL	
COUNTERS=LIST	
,COUNTERLIST=*counterlist*	Required with COUNTER=LIST
SUBPOOL=0	**Default:** SUBPOOL=0
SUBPOOL=*subpool*	
FSDADDRESS=*fsdaddress*	
FSDLENGTH=*fsdlength*	

Syntax	Description
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,MF=S	**Default**: MF=S
,MF=(L,*mfctrl*)	*list addr*: RS-type address or register (1) - (12).
,MF=(L,)	
,MF=(L,*r*,0D)	
,MF=(E,)	
,MF=(E,)	
,MF=(E,)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IRDFSD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,DEVICE=ALL
,DEVICE=SINGLE
An optional parameter that indicates the one or more devices for which statistics should be returned.

DEVICE=ALL
Return statistics for all devices.

DEVICE=SINGLE
Return statistics for a single device.

DEVICENUMBER=*devicenumber*
A required input parameter if DEVICE=SINGLE is specified. It identifies the device number of the FICON switch device to be interrogated.

To code: Specify the RS-type address or address in register (2)-(12), of a 4 byte field.

,COUNTERS=DEFAULT
,COUNTERS=ALL
COUNTERS=LIST
An optional parameter that indicates the set of counters to be returned.

COUNTERS=DEFAULT
Specifies that the default set of counters is to be returned.

COUNTERS=ALL
Specifies that all supported counters should be returned.

COUNTERS=LIST
Specifies that the list of counters to be returned has been supplied.

COUNTERLIST=*counterlist*
A required input parameter if COUNTER=LIST is specified. It contains the address that specifies a list of Statistical Counter Control Words to return.

To code: Specify the RS-type address or address in register (2)-(12), of a 4 byte field.

The number of Control Words cannot exceed 60.

See the IHAFSD macro for further information about the definition of the statistical counter list (SCCW) and the counters available.

,SUBPOOL=0
Subpool=*subpool*Syntax
A optional input parameter that specifies the subpool to be used to obtain the storage for the FICON Switch Dat (FSD). The default is 0.

To code: Specify the RS-type address or address in register (2)-(12) of the 1 byte field containing the subpool.

The returned information is mapped in macro IHAFSD.

,FSDADDRESS=*fsdaddress*
A required input parameter which contains the address of the storage mapped in IHAFSD.

The mapping macro IHAFSD can be found in SYS1.MODGEN.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,FSDLENGTH=*fsdlength*
A required input parameter which contains the length of the storage mapped in IHAFSD.

To code: Specify the RS-type address of a 4-byte field, or register (2)-(12).

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM suggests that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=S
,MF=(L,*mfctrl*)
,MF=(L,*mfctrl*,*mfattr*)
,MF=(L,*mfctrl*,0D)
,MF=(E,*mfctrl*)
,MF=(E,*mfctrl*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. The execute form of the macro stores the parameters into the storage area defined by xmfctrl and provides full syntax checking with the default setting.

,xmfctrl
This is a required keyword that specifies a storage area for the parameter list. This can be an RS-type address or an address in register (1)-(12).

,xmfattr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return codes

When the IRDFSD macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return codes:

Table 106. Return Codes for IRDFSD macro

Hexadecimal Return Code	Meaning
00	**Meaning**: Successful completion. Data returned for all FICON switches.
04	**Meaning**: No FICON switch devices found. No FSD area was obtained.
08	**Meaning**: I/O errors occurred. Some switches did not return data.
0C	**Meaning**: I/O errors occurred. No data obtained for any devices.
10	**Meaning**: Unexpected error.

Reason codes

The following table identifies the hexadecimal return and reason codes:

Table 107. Return and Reason Codes for IRDFSD macro

Hexadecimal Return Code	Return Code	Meaning
00000001	04	Required module unavailable. The request could not be processed.
00000002	04	The server task is not active. The request could not be processed.
00000024	04	FICON switch statistics disabled.
00000003	08	Caller is not APF authorized.
00000008	08	IRDVFSD does not recognize the request type.
00000021	08	Specified device is not a FICON switch.
00000022	08	Specified device is not online.
00000023	08	IOSVFSD does not recognize the request type.
00000004	0C	Recovery could not be established.
00000005	0C	POST failed.
00000006	0C	Recovery was entered.
00000025	0C	I/O error.
00000026	0C	UCBSCAN failure.
00000027	0C	UCBLOOK failure.
00000028	0C	Server task recovery entered.

Chapter 123. IRDFSDU — FICON switch data update services

Description

The FICON Switch Data macro update service is used to update statistical counters from FICON switch devices.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or Supervisor state or any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit addressing mode.
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None

Restrictions

- No locks may be held
- Must not be in an environment that would prevent EXCP from being issued
- Must be authorized

Input register information

Register
Contents

0–15
Undefined

Before issuing the IRDFSDU macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason Code

1
Used as a work register by the system.

2-13
Restored

14
Used as a work register by the system.

15
Return code

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The IRDFSDU macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IRDFSDU.
IRDFSDU	
ƀ	One or more blanks must follow IRDFSDU.
,TOLERANCE=*tolerance*	*tolerance*RS-type address or address in register (2) — (12).
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default**: PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=1	
,MF=S	**Default**: MF=S
,MF=(L,*mfctrl*)	

Syntax	Description
,MF=(L,*mfctrl,mfattr*)	
,MF=(L,*mfctrl*,0D)	
,MF=(E,*mfctrl*)	
,MF=(E,COMPLETE)	

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the IRDFSDU macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,TOLERANCE=*tolerance*
A required input parameter which contains the input tolerance for up-to-date test of last update performed.

To code: Specify the RS-type address or address in register (2)-(12), of an 8 byte field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1

,MF=S
,MF=(L,*mfctrl*)
,MF=(L,*mfctrl*,*mfattr*)
,MF=(L,*mfctdrl*,0D)
,MF=(E,*mfctrl*)
,MF=(E,*mfctrl*,COMPLETE)

An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. The execute form of the macro stores the parameters into the storage area defined by *mfctrl* and provides full syntax checking with the default setting.

,mfctrl
This is a required keyword that specifies a storage area for the parameter list. This can be an RS-type address or an address in register (1)-(12).

,mfattr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *mfattr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

ABEND codes

None.

Return codes

When the IRDFSDU macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes:

Table 108. Return Codes for IRDFSDU macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00		**Meaning**: Successful completion. Data updated for all FICON switches.
04	2	Warning **Meaning**: The server task is not available
04	24	Environmental Error **Meaning**: FICON switch statistics disabled
08	1	**Meaning**: The subpool provided by the caller is not in common storage.
08	2	**Meaning**: The switch device provided by the caller is not in the Switch Table.

Table 108. Return Codes for IRDFSDU macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning**
20	1	**Meaning**: An ESTAE could not be established.

Reason codes

The following table identifies the hexadecimal return and reason codes:

Table 109. Return and Reason Codes for IRDFSD macro		
Hexadecimal Return Code	**Return Code**	**Meaning**
00000001	04	Required module was not found. The request could not be processed.
00000002	04	The server task is unavailable. The request could not be processed.
00000024	04	FICON switch statistics disabled.
00000003	08	Caller is not APF authorized.
00000008	08	IRDVFSD does not recognize the request type.
00000023	08	IOSVFSD does not recognize the request type.
00000004	0C	Recovery could not be established.
00000005	0C	POST failed.
00000006	0C	Recovery was entered.
00000025	0C	I/O error.
00000026	0C	UCBSCAN failure.
00000027	0C	UCBLOOK failure.
00000028	0C	Server task recovery entered.

Chapter 124. ISGADMIN — Global resource serialization administration service

Description

Interface for Global Resource Serialization Administration

The GRS Administration service routine is given control from the ISGADMIN macro to:

- Change maximum ENQ limits for a specific address space.
- Move an ENQ waiter to a different position in the request queue and to optionally change its control type from exclusive to shared.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be authorized, although any one of the following attributes is sufficient: • Supervisor State • Key 0-7 • APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN Note: The updated ENQ limit is updated for the home address space.
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The control parameters must be in the same key as the caller.

Programming requirements

The caller must include the ISGYCON macro to get the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs)

This macro supports multiple versions. Some keywords are unique to certain versions. See the PLISTVER parameter description.

Input register information

Before issuing the ISGADMIN macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

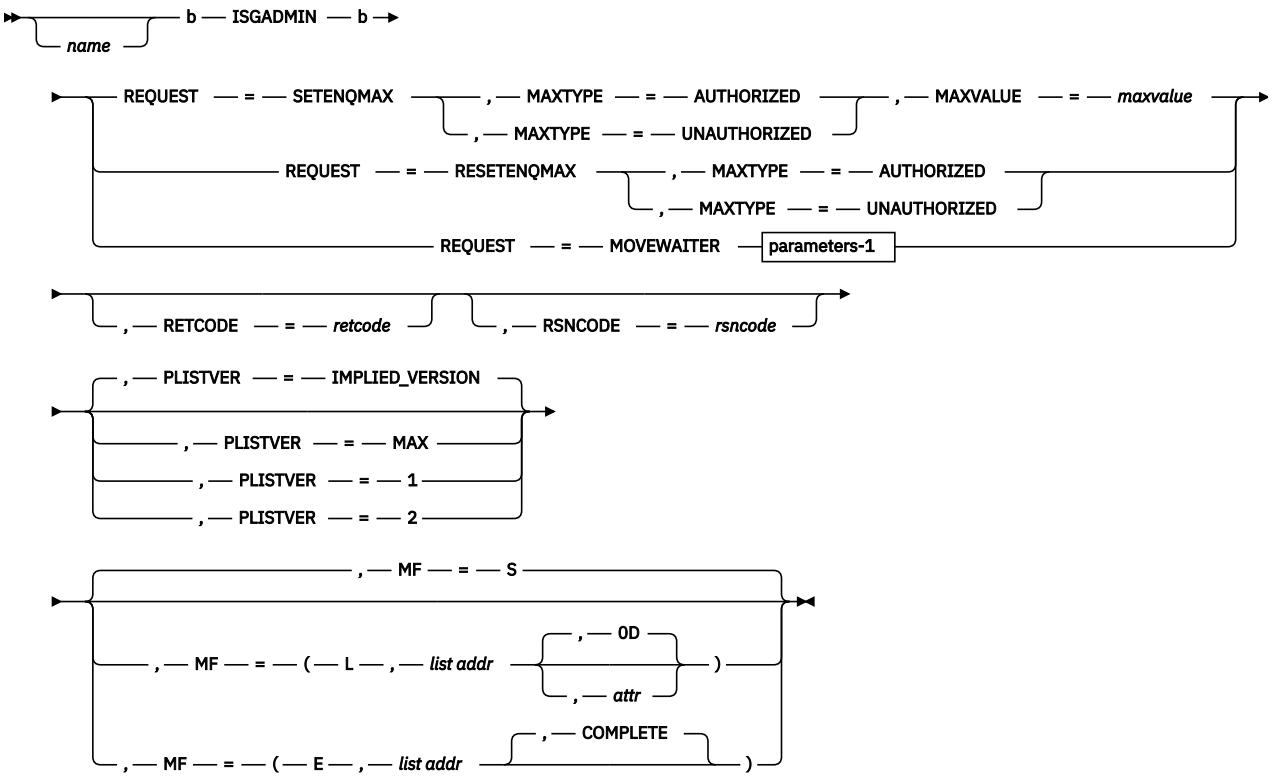
Performance implications

None.

Syntax

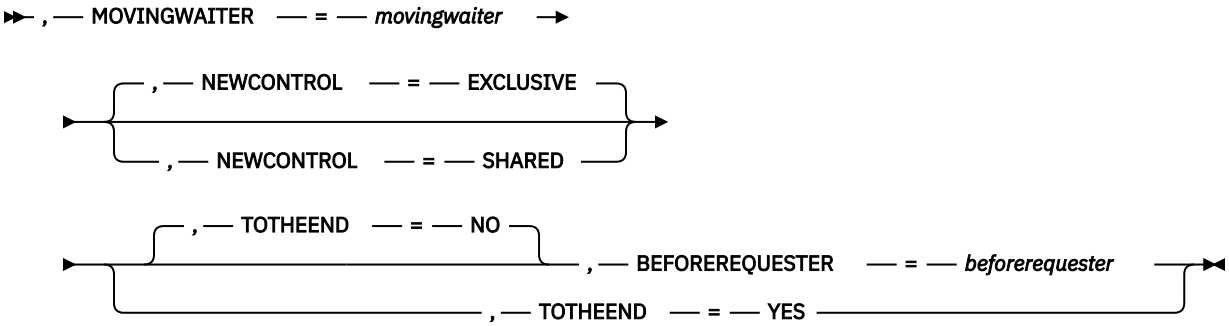
Note:

main diagram



Note:

parameters-1



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the ISGADMIN macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,BEFOREREQUESTER=*beforerequester*
When TOTHEEND=NO and REQUEST=MOVEWAITER are specified, a required input parameter that is an ENQToken identifying the ENQ request that the MovingWaiter request precedes.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,MAXTYPE=AUTHORIZED
,MAXTYPE=UNAUTHORIZED
When REQUEST=SETENQMAX is specified, a required parameter.

,MAXTYPE=AUTHORIZED
Indicates a request to update the maximum ENQ limit for authorized requesters.

,MAXTYPE=UNAUTHORIZED
Indicates a request to update the maximum ENQ limit for unauthorized requesters.

,MAXTYPE=AUTHORIZED
,MAXTYPE=UNAUTHORIZED
When REQUEST=RESETENQMAX is specified, a required parameter.

,MAXTYPE=AUTHORIZED
Indicates a request to reset the maximum ENQ limit for authorized requesters.

,MAXTYPE=UNAUTHORIZED
Indicates a request to reset the maximum ENQ limit for unauthorized requesters.

,MAXVALUE=*maxvalue*
When REQUEST=SETENQMAX is specified, a required input parameter that is the requested value of the new maximum ENQ limit. The specified value must be greater than or equal to the absolute minimum described in ISGYCON, and up to 2?1-1 (2147483647).

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MOVINGWAITER=*movingwaiter*
When REQUEST=MOVEWAITER is specified, a required input parameter that is an ENQToken identifying the ENQ waiter.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,NEWCONTROL=EXCLUSIVE
,NEWCONTROL=SHARED
When REQUEST=MOVEWAITER is specified, an optional parameter. The default is NEWCONTROL=EXCLUSIVE.

,NEWCONTROL=EXCLUSIVE
Indicates that the requester represented by the MovingWaiter ENQToken should have its control remain Exclusive.

,NEWCONTROL=SHARED
Indicates that the request represented by the MovingWaiter ENQToken should have its control become Shared.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
,PLISTVER=2
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, which supports all parameters except those specifically referenced in higher versions.
- **2**, which supports both the following parameters and those from version 1:
  - BEFOREREQUESTER
  - MOVINGWAITER
  - NEWCONTROL
  - TOTHEEND

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1, or 2

REQUEST=SETENQMAX
REQUEST=RESETENQMAX
REQUEST=MOVEWAITER
A required parameter that indicates the type of ISGADMIN request.

REQUEST=SETENQMAX
Indicates a request to change the ENQ maximum for the home address space.

REQUEST=RESETENQMAX
Indicates a request to reset the ENQ maximum for the home address space back to the system default.

REQUEST=MOVEWAITER
Indicates a request to move an ENQ waiter to a different position in the request queue and to optionally change its control type through the NEWCONTROL keyword.

This request requires a version 2 parameter list.

Note: This function is intended to only be used by third party serialization products. Its misuse can result in deadlocks, incorrect serialization or loss of data integrity. The MOVEWAITER, TOTHEEND, and BEFOREREQUESTER keywords specify which requester should be moved and where to move it. The waiter will only be moved under the following conditions:

- The MOVINGWAITER:
  - Has a requested disposition of Exclusive
  - Is not currently an owner of the resource
  - Cannot result in any new owners as a result of the move
  - Must be waiting for the same resource as the BEFOREREQUESTER (if specified)
- The resource is NOT global or STEP in scope. Note that in GRS=NONE mode, the final scope can be SYSTEMS or SYSTEM. When in other GRS modes the scope must be SYSTEM.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,TOTHEEND=NO
,TOTHEEND=YES
When REQUEST=MOVEWAITER is specified, an optional parameter. The default is TOTHEEND=NO.

,TOTHEEND=NO
Indicates that the requester represented by the MovingWaiter ENQToken should be moved to a position specified through the BEFOREREQUESTER keyword.

,TOTHEEND=YES
Indicates that the request represented by the MovingWaiter ENQToken should be moved to the end of the request queue.

ABEND codes

None

Return and reason codes

When the ISGADMIN macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro ISGYCON provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 110. Return and Reason Codes for the ISGADMIN Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
00	—	**Equate Symbol**: ISGADMINRc_OK **Meaning**: ISGADMIN request successful. **Action**: None required.
04	—	**Equate Symbol**: ISGADMINRc_Warn **Meaning**: Warning **Action**: Refer to action under the individual reason code.
04	xxxx0401	**Equate Symbol**: ISGADMINRsn_ENQMaxValueLow **Meaning**: For REQUEST=SETENQMAX. The specified maximum is less than or equal to the current system-wide maximum. This space-specific maximum has been set but has no immediate effect. **Action**: Ensure the specified MaxValue is accurate. If not, reissue the ISGADMIN service with a higher value.
04	xxxx0402	**Equate Symbol**: ISGADMINRsn_ResetENQMaxIgnored **Meaning**: For REQUEST=RESETENQMAX. The home address space did not have a specific maximum for that type of requester. **Action**: Ensure that the reset was desired, and issued for the appropriate requester type, authorized or unauthorized. Reissue the service with the correct requester type if appropriate.
08	—	**Equate Symbol**: ISGADMINRc_ParmError **Meaning**: ISGADMIN request specified parameters in error. **Action**: Refer to action under the individual reason code.
08	xxxx0801	**Equate Symbol**: ISGADMINRsn_BadPlistAddress **Meaning**: Unable to access parameter list. **Action**: Check that the parameter list is addressable. If in AR-mode, check that the ALET of the parameter list is correct. Note that if this macro is issued in AR-mode, SYSSTATE ASCENV=AR must be issued before this macro. Ensure that the storage is in the same key as the caller.
08	xxxx0802	**Equate Symbol**: ISGADMINRsn_BadPlistALET **Meaning**: Bad parameter list ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Make sure that the ALET of the parameter list is valid. Its access register may not have been set up properly.

Table 110. Return and Reason Codes for the ISGADMIN Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0803	**Equate Symbol**: ISGADMINRsn_BadPlistVersion **Meaning**: Bad parameter list version number. The ISGADMIN parameter list version is greater than the version supported by GRS on the current system or the ISGADMIN parameter list version is lower than the minimum required for parameters that were specified. **Action**: Check for possible storage overlay of the parameter list. Retry the request with the correct version number. Verify that your program was assembled with the correct macro library for the release of MVS on which your program is running.
08	xxxx0804	**Equate Symbol**: ISGADMINRsn_ReservedFieldNotNull **Meaning**: A reserved field in the parameter list is non-zero. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0805	**Equate Symbol**: ISGADMINRsn_BadRequest **Meaning**: Bad REQUEST parameter. **Action**: IBM suggests that the ISGADMIN macro is used when invoking the ISGADMIN service.
08	xxxx0806	**Equate Symbol**: ISGADMINRsn_ENQMaxValueTooLow **Meaning**: For REQUEST=SETENQMAX. The specified maximum is less than the smallest allowable maximum. **Action**: Check the smallest allowable maximum in macro ISGYCON. Reissue the ISGADMIN service with a higher value.
08	xxxx0807	**Equate Symbol**: ISGADMINRsn_BadMovingWaiterAddress **Meaning**: For REQUEST=MOVEWAITER. Unable to access the MovingWaiter ENQToken. **Action**: Make sure that the MovingWaiter ENQToken is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller.
08	xxxx0808	**Equate Symbol**: ISGADMINRsn_BadMovingWaiterAlet **Meaning**: For REQUEST=MOVEWAITER. Bad MovingWaiter ENQToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Make sure that the ALET of the MovingWaiter ENQToken is valid. Its access register may not have been set up properly.

Table 110. Return and Reason Codes for the ISGADMIN Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0809	**Equate Symbol**: ISGADMINRsn_BadMovingWaiter **Meaning**: For REQUEST=MOVEWAITER. The specified MovingWaiter ENQToken does not represent an ENQ on the current system. **Action**: Make sure that the specified MovingWaiter ENQToken is from a previous request that has not been subsequently released.
08	xxxx080A	**Equate Symbol**: ISGADMINRsn_BadBeforeRequesterAddress **Meaning**: For REQUEST=MOVEWAITER. Unable to access the BeforeRequester ENQToken. **Action**: Make sure that the BeforeRequester ENQToken is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller.
08	xxxx080B	**Equate Symbol**: ISGADMINRsn_BadBeforeRequesterAlet **Meaning**: For REQUEST=MOVEWAITER. Bad BeforeRequester ENQToken The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Make sure that the ALET of the BeforeRequester ENQToken is valid. Its access register may not have been set up properly.
08	xxxx080C	**Equate Symbol**: ISGADMINRsn_BadBeforeRequester **Meaning**: For REQUEST=MOVEWAITER. The specified BeforeRequester ENQToken does not represent an ENQ on the current system. **Action**: Make sure that the specified BeforeRequester ENQToken is from a previous request that has not been subsequently released.
08	xxxx080D	**Equate Symbol**: ISGADMINRsn_SameRequester **Meaning**: For REQUEST=MOVEWAITER. The same ENQToken was specified for both MovingWaiter and BeforeRequester. **Action**: Make sure that the ENQTokens are distinct.
08	xxxx080E	**Equate Symbol**: ISGADMINRsn_InconsistentResource **Meaning**: For REQUEST=MOVEWAITER. The specified MovingWaiter and BeforeRequester ENQTokens do not represent ENQ requests for the same resource. **Action**: Make sure that the ENQTokens specified are against the same resource.

Table 110. Return and Reason Codes for the ISGADMIN Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx080F	**Equate Symbol**: ISGADMINRsn_BadScope **Meaning**: For REQUEST=MOVEWAITER. The resource associated with the specified MovingWaiter and BeforeRequester ENQTokens is not a valid local resource. The resource cannot be a global or a STEP resource. Note that in GRS=NONE mode, an acceptable local resource can have a final scope of SYSTEMS or SYSTEM. When in other GRS modes, the final scope can only be SYSTEM. **Action**: Make sure that the ENQTokens specified are against a valid local resource.
08	xxxx0810	**Equate Symbol**: ISGADMINRsn_BadControl **Meaning**: For REQUEST=MOVEWAITER. The specified MovingWaiter ENQToken represents a requester of shared control. **Action**: Make sure that the MovingWaiter ENQToken represents a requester of exclusive control.
08	xxxx0811	**Equate Symbol**: ISGADMINRsn_CannotMoveOwner **Meaning**: For REQUEST=MOVEWAITER. The specified MovingWaiter ENQToken represents a requester that owns the resource. **Action**: Make sure that the MovingWaiter ENQToken specified is for a waiting requester.
08	xxxx0812	**Equate Symbol**: ISGADMINRsn_AlreadyBeforeRequester **Meaning**: For REQUEST=MOVEWAITER, TOTHEEND=NO, BEFOREREQUESTER=. The specified MovingWaiter ENQToken represents a requester that is already queued just before the requester represented by the BeforeRequester ENQToken. The control was not changed. **Action**: Make sure that the MovingWaiter and BeforeRequester ENQTokens represent the correct requesters and that the queue is as expected.
08	xxxx0813	**Equate Symbol**: ISGADMINRsn_CannotMoveBeforeOwner **Meaning**: For REQUEST=MOVEWAITER. If granted, the requester represented by the MovingWaiter ENQToken would have become the owner of the resource because it would precede an owner. **Action**: Make sure that the BeforeRequester ENQToken represents a waiting requester.

Table 110. Return and Reason Codes for the ISGADMIN Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0814	**Equate Symbol**: ISGADMINRsn_CannotMoveAfterSharedOwner **Meaning**: For REQUEST=MOVEWAITER, NEWCONTROL=SHARED. If granted, the requester represented by the MovingWaiter ENQToken would have become the owner of the resource because it would immediately follow a shared owner. **Action**: Make sure that the BeforeRequester ENQToken represents the requester that the moving waiter should precede.
08	xxxx0815	**Equate Symbol**: ISGADMINRsn_CannotMakeAnotherOwner **Meaning**: For REQUEST=MOVEWAITER. If granted, one or more requesters queued after the one represented by the MovingWaiter ENQToken would have become the owner of the resource. **Action**: Make sure that the MOVEWAITER request would not make any other waiting requesters the owner of the resource.
08	xxxx0816	**Equate Symbol**: ISGADMINRsn_AlreadyLastRequester **Meaning**: For REQUEST=MOVEWAITER, TOTHEEND=YES. The requester represented by the MovingWaiter ENQToken is already at the end of the request queue. **Action**: Make sure that the MovingWaiter ENQToken represents a requester at the correct position and that the request queue is as expected.
08	xxxx0817	**Equate Symbol**: ISGADMINRsn_CannotMoveMasidUser **Meaning**: For REQUEST=MOVEWAITER. The MovingWaiter ENQToken represents a MASID user. **Action**: Make sure that the MovingWatier ENQToken does not represent a MASID user.
08	xxxx0818	**Equate Symbol**: ISGADMINRsn_MasidControlConflict **Meaning**: For REQUEST=MOVEWAITER, NEWCONTROL=SHARED. The requester represented by the MovingWaiter ENQToken would create a bad MASID environment since a shared owner of the resource is a convert-to-exclusive MASID target. **Action**: Make sure that the requester represented by the MovingWaiter ENQToken would not need to move in the midst of a MASID convert-to-exclusive environment or that the moved requester could maintain a control of Exclusive.
0C	—	**Equate Symbol**: ISGADMINRc_EnvError **Meaning**: ISGADMIN request has an environment error. **Action**: Refer to action under the individual reason code.

Table 110. Return and Reason Codes for the ISGADMIN Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
0C	xxxx0C01	**Equate Symbol**: ISGADMINRsn_NotAuthorized **Meaning**: An unauthorized caller invoked the ISGADMIN service. **Action**: An ISGADMIN caller must be authorized.
0C	xxxx0C02	**Equate Symbol**: ISGADMINRsn_FRRHeld **Meaning**: The caller issued ISGADMIN when an FRR was established. **Action**: Avoid issuing ISGADMIN when using functional recovery routines.
0C	xxxx0C03	**Equate Symbol**: ISGADMINRsn_LockHeld **Meaning**: A lock was held upon entry. No locks may be held when calling ISGADMIN. **Action**: Avoid using ISGADMIN when locks are held.
0C	xxxx0C04	**Equate Symbol**: ISGADMINRsn_SrbMode **Meaning**: SRB mode. **Action**: SRB mode is not supported.
0C	xxxx0C05	**Equate Symbol**: ISGADMINRsn_NotEnabled **Meaning**: Not Enabled. **Action**: Avoid using ISGADMIN when not enabled.
0C	xxxx0C06	**Equate Symbol**: ISGADMINRsn_QueueDamage1 **Meaning**: The GRS resource queue structure for the target resource is damaged. Further processing against the queue is not allowed. **Action**: Prevent any further processing against the target resource.
0C	xxxx0C07	**Equate Symbol**: ISGADMINRsn_QueueDamage2 **Meaning**: The GRS resource queue structure for the target resource is damaged. Further processing against the queue is not allowed. **Action**: Prevent any further processing against the target resource.
10	—	**Equate Symbol**: ISGADMINRc_CompError **Meaning**: Component Error **Action**: Contact the IBM Support Center. Provide the reason code which contains diagnostic data.

Examples

```
    * ********************************************************************
    *  Set the unauthorized ENQ maximum for the home address space
```

```
* ********************************************************************

         ISGADMIN REQUEST=SETENQMAX,                                  X
            MAXTYPE=UNAUTHORIZED,MAXVALUE=MYVALUE,                 X
            RETCODE=MYRC,RSNCODE=MYRSN

* ********************************************************************
*  Reset the unauthorized ENQ maximum of the home address space
* ********************************************************************

         ISGADMIN REQUEST=RESETENQMAX,                                X
            MAXTYPE=UNAUTHORIZED,                                  X
            RETCODE=MYRC,RSNCODE=MYRSN
```

```
* ********************************************************************
*  Move an ENQ Waiter
* ********************************************************************

          ISGADMIN REQUEST=MOVEWAITER,                                 X
             MOVINGWAITER=mywaiterENQToken,                            X
             TOTHEEND=NO,                                              X
               BEFOREREQUESTER=mybeforerequesterENQToken,              X
             RETCODE=MYRC,RSNCODE=MYRSN
```

For more information about global resource serialization, see *z/OS MVS Planning: Global Resource Serialization*.

Chapter 125. ISGECA — GRS enhanced contention analysis service

Description

Use the ISGECA service to obtain waiter and blocker information for global resource serialization (GRS) component managed resources. GRS resource waiter/blocker information can be obtained for a specific system within the current sysplex, or for all the systems operating in the current sysplex.

A GRS resource is considered relevant to an ISGECA request if that resource currently has waiters and blockers associated with it. For a given relevant GRS resource, ISGECA returns the following types of information:

Waiter
The longest waiting unit of work for that resource, and the top (longest) blocking unit of work for that waiter. Further general information about the resource and the numbers of resource owners and waiters is also reported.

Blocker
The longest blocking unit of work for that resource. Further general information about the resource and the numbers of resource owners and waiters is also reported.

ISGECA returns information for as many relevant GRS resources as is specified by the COUNT parameter. All reported resource information is collected into a virtual storage buffer specified by the RIBOUT parameter. Reported information is formatted according to RIB and RIBE DSECTs, available from syslib member ISGRIB. See WAITER and BLOCKER descriptions under the “REQUEST=WAITER” on page 1200 parameter for the specific RIBOUT buffer area format. For precise descriptions of resource, waiter and blocker information reported, see "RIB Heading Information" in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

ISGECA reports on relevant resources as they are encountered in the system's GRS resource management data infrastructure. The order of reported resources in the RIBOUT area is unpredictable, and implies no suggestion of one resource having greater waiter/blocker considerations than any other reported on resource.

The ISGECA service might be unable to report any waiter or blocker information for some sysplex systems, in some invocation cases, for a variety of reasons. In the event that this occurs, ISGECA reports the system names of systems not included in the report, and the reason for not including those systems, in the NOTINCL output area. The description for parameter NOTINCL explains the output area format and reason codes associated with it.

Note: The 476-byte (or X'1DC') parameter list constructed by ISGECA and passed to its service routine MUST reside in common area subpool 231. This requirement has significant implications on the use of the various macro format (MF) options. For more information about this parameter list requirement, see “Programming requirements” on page 1196.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state. Zero PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit

Environmental factor	Requirement
ASC mode:	Primary, Secondary, access register (AR), or Home
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

1. The parameter list constructed by ISGECA and passed to its service routine MUST reside in common area subpool 231. This has the following implications:
   - For assembler standard format invocations (i.e., MF=(S)), the invoking program code must reside in subpool 231, as an inline parameter list is generated.
   - For PL/X standard format invocations (i.e., MF(S)), the invoker's dynamic area must reside in subpool 231, as a dynamic area declare for the parameter list is generated.
   - Similarly, for list format invocations (i.e., MF=(L,*xxx*)), if the resulting declared parameter list resides in the program's dynamic storage area, then this storage must be obtained from subpool 231.

     If the resulting list format parameter list declare is a PL/X based construct, then the program may substantiate the based construct via an allocated subpool 231 address for subsequent execute format (i.e., MF=(E,*xxx*)) invocations.
   - For execute format invocations (i.e., MF=(E,*xxx*)), the specified parameter list must reside in common area subpool 231.

   The parameter list must be 476 (or X'1DC') bytes in length.
2. PL/X invokers must include syslib members CVT and ISGGVT.
3. Include syslib member ISGRIB for RIB and RIBE DSECT mappings. These DSECTs precisely describe formatted areas in the RIBOUT area.
4. ISGECA service return and reason codes can be retrieved from the ISGECA parameter list area, as an alternative to coding the RETCODE and RSNCODE parameters. These results appear in the parameter list as follows:
   - Return code: 2-byte value at offset 60 (or X'3C').
   - Reason code: 2-byte value at offset 62 (or X'3E').
5. The ISGECA service requires a specific system service or release level to function successfully. The ISGECA macro expansion performs before any other tests and calling the service routine, verifying the system has this function enabled.

Restrictions

None

Input register information

Before issuing the ISGECA macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Before issuing the ISGECA macro, the caller does not have to place any information into any access register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Unchanged

1
Unpredictable

2-13
Unchanged

14
Unpredictable

15
Unchanged

When control returns to the caller, the ARs contain:

Register
Contents

0
Unchanged

1
Unpredictable

2-13
Unchanged

14
Unpredictable

15
Unchanged

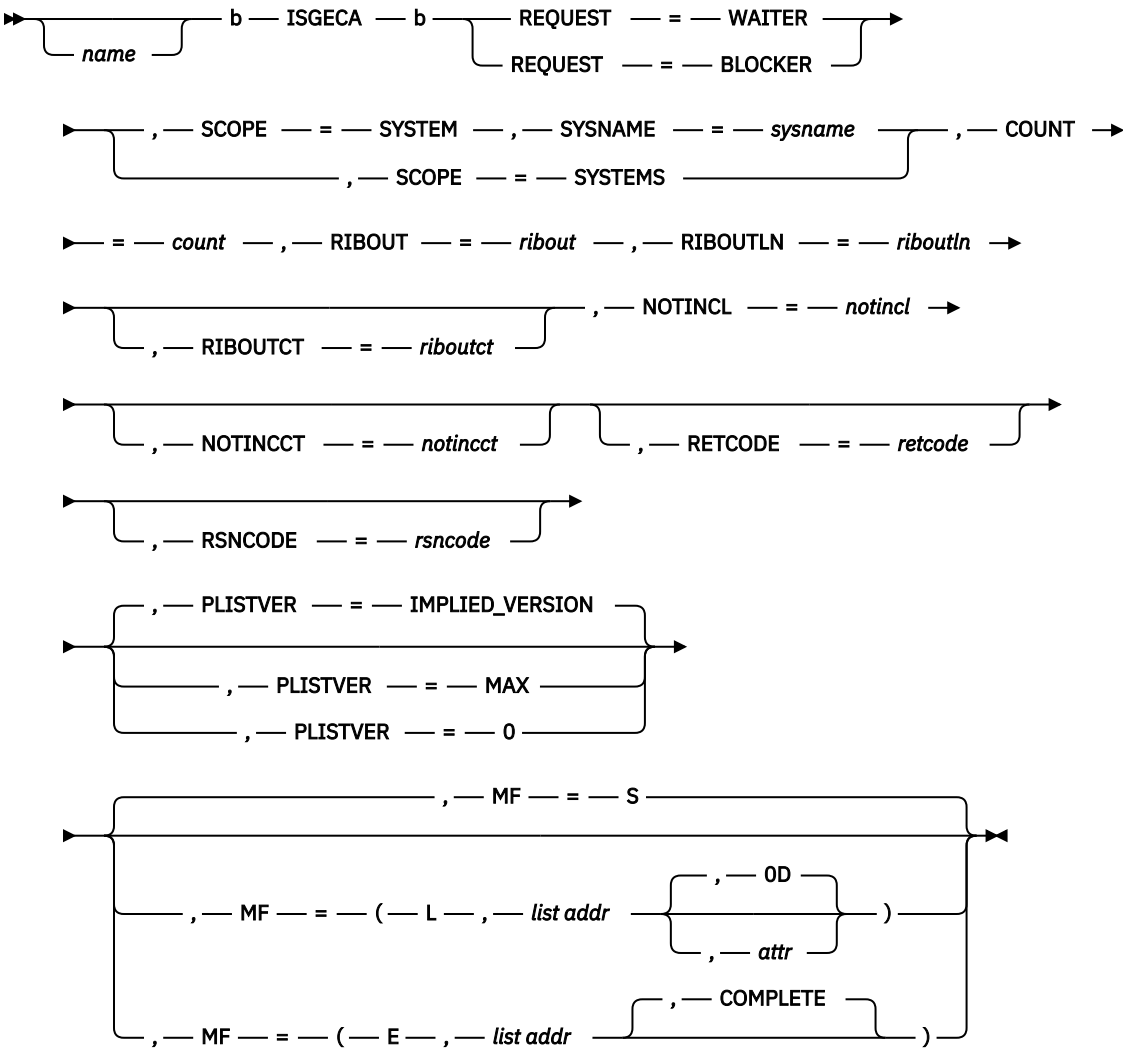
Performance implications

None.

Syntax

Note:

main diagram



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the ISGECA macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,COUNT=*count*
A required input parameter describing the maximum number of relevant resources to be reported on by this ISGECA invocation. The maximum value that can be specified with the COUNT parameter is 99.

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOTINCCT=*notincct*
An optional output parameter, to contain the number of systems reported in the NOTINCL area. Alternatively, this number of NOTINCL entries can be obtained from the 2-byte parameter list field at offset 58 (or X'3A').

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,NOTINCL=*notincl*
A required input parameter that contains the address of a virtual storage output area to contain the list of systems for which RIBs and RIBEs are not included in the RIBOUT area. The NOTINCL area must begin on a doubleword boundary, and must reside within common storage subpool 231.

The length of the NOTINCL area, in bytes, must, minimally, be the number of systems currently executing in the sysplex multiplied by 10 (or X'0A'). The format of the NOTINCL area is as follows:

```
+--------------------------+
| System name | Reason Code |
| System name | Reason Code |
| System name | Reason Code |
+--------------------------+
```

Each system name and reason code pair potentially reflects a system not included in waiter/blocker data returned in the RIBOUT area. The number of systems reported on in the NOTINCL area is returned in the NOTINCCT output parameter value.

Each NOTINCL system name field is an 8-byte field, and each reason code entry is a 2-byte field. Reason codes for the NOTINCL area are independent of ISGECA service invocation reason codes, and are only meaningful when the ISGECA return code is 4 or less. The NOTINCL reason codes and meanings are as follows:

Hex Reason Code
Meaning

0000
Ignore this NOTINCL area entry, including the system name value specified.

0001
The system described by the system name field is cannot process the ISGECA service.

0002
The system described by the system name field was not found to be participating in the current sysplex.

0003
The system described by the system name field did not respond to an XCF request to gather ISGECA report information.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

REQUEST=WAITER
REQUEST=BLOCKER
A required parameter that indicates the type of ISGECA request.

REQUEST=WAITER
When you specify WAITER, the longest waiters and top blockers for each relevant resource are returned. For REQUEST=WAITER, the RIBOUT buffer area is formatted as follows:

```
+-------------------+
| RIB | RIBE | RIBE |
| RIB | RIBE | RIBE |
| RIB | RIBE | RIBE |
+-------------------+
```

The number of RIBs collected in the RIBOUT area is returned in the RIBOUTCT parameter variable. Each RIB/RIBE/RIBE trio reports on the following:

- The RIB describes general information about the resource, including the QNAME, minor name and the numbers of waiters and blockers.
- The first RIBE describes the top blocking unit of work for this resource.
- The second RIBE describes the longest waiting unit of work for this resource.

REQUEST=BLOCKER
When you specify BLOCKER, the top blockers for each relevant resource is returned. For REQUEST=BLOCKER, the RIBOUT buffer area is formatted as follows:

```
+------------+
| RIB | RIBE |
```

```
| RIB | RIBE |
| RIB | RIBE |
+------------+
```

The number of RIBs collected in the RIBOUT area is returned in the RIBOUTCT parameter variable. Each RIB/RIBE pair reports on the following items:

- The RIB describes general information about the resource, including the QNAME, minor name and the numbers of waiters and blockers.
- The RIBE describes the top blocking unit of work for this resource.

,RETCODE=*retcode*
An optional output parameter that will contain the return code.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RIBOUT=*ribout*
A required input parameter that contains the address of the virtual storage output area for this request. The RIBOUT area must reside in the invoker's primary address space, and contains the ISGECA report of RIBs and RIBEs for the request.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,RIBOUTCT=*riboutct*
An optional output parameter, to contain the number of RIBs collected in the RIBOUT area. Alternatively, this number of RIBs can be obtained from the 2-byte parameter list field at offset 56 (or X'38').

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field.

,RIBOUTLN=*riboutln*
A required input parameter describing the length, in bytes, of the RIBOUT virtual storage area.

The length of the RIBOUT area must be large enough to accomodate the maximum size ISGECA report for the request, and therefore must be of a magnitude that facilitates the COUNT parameter value and RIB/RIBE DSECT mapping sizes. Depending on the ISGECA request type, this relationship between these parameter values and DSECT sizes can be expressed as follows:

Waiter:
RIBOUTLN parameter value must equal or exceed the COUNT parameter value multiplied by 392 (or X'188').

Blocker:
RIBOUTLN parameter value must equal or exceed the COUNT parameter value multiplied by 344 (or X'158').

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,RSNCODE=*rsncode*
An optional output parameter that will contain the reason code.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SCOPE=SYSTEM
,SCOPE=SYSTEMS
A required parameter that indicates the request scope.

,SCOPE=SYSTEM
ISGECA is to only report on blockers and, potentially, waiters currently executing on a specific system within the current GRS complex.

,SCOPE=SYSTEMS
ISGECA is to report on blockers and, potentially, waiters across all of the systems in the current sysplex complex.

,SYSNAME=*sysname*
When SCOPE=SYSTEM is specified, a required input parameter string containing the system name of the single system on which ISGECA is to report.

SYSNAME is required when you specify SCOPE=SYSTEM. SYSNAME is not valid for SCOPE=SYSTEMS.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

ABEND codes

None.

Return and reason codes

When the ISGECA macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes. IBM support personnel may request the entire reason code, including the **xxxx** value.

Table 111. Return and Reason Codes for the ISGECA Macro

Return Code	Reason Code	Meaning and Action
00	–	Successful completion. The RIBOUT virtual storage area contains the waiter or blocker output report, and the NOTINCL virtual storage area describes system names and reason codes for systems not reported on in the RIBOUT area.
04	–	Request completed with exceptional circumstances.
04	*xxxx*0000	**Meaning:** The ISGECA service could not communicate with some systems in the sysplex, so the returned data is incomplete. This implies that such systems are non-responsive, such as when a system has been reset but has not been removed from the sysplex. **Action:** No suggested program action.
04	*xxxx*0001	**Meaning**: Not all systems in the current GRS complex that are relevant to the ISGECA request are participating in the current sysplex. **Action**: No suggested program action. The RIBOUT virtual storage area contains the waiter or blocker output report, and the NOTINCL virtual storage area describes system names and reason codes for systems not reported on in the RIBOUT area.
08	–	Request failed.
08	*xxxx*0000	**Meaning**: A GRS internal error occurred and the request could not be completed. **Action**: No suggested program action.

Table 111. Return and Reason Codes for the ISGECA Macro (continued)		
Return Code	**Reason Code**	**Meaning and Action**
08	*xxxx*0001	**Meaning**: The ISGECA service routine was unable to obtain storage necessary to process the request. **Action**: Consider reducing the COUNT and RIBOUTLN parameter values to decrease the total number of resources to be reported, and re-invoke ISGECA. Alternatively, if this was a SCOPE(SYSTEMS) request, consider re-inovking ISGECA with SCOPE(SYSTEM) to, potentially, reduce the total number of resources to be reported.
08	*xxxx*0003	**Meaning**: A GRS internal error occurred and the request could not be completed. **Action**: No suggested program action.
08	*xxxx*0004	**Meaning**: The sysplex is in the process of migrating to GRS STAR mode, and therefore cannot process the request at this time. **Action**: Iteratively retry the ISGECA invocation, waiting a few seconds between attempts.
08	*xxxx*00FD	**Meaning**: The maximum number of relevant resources to be reported on, as specified by parameter COUNT, exceeds the service maximum value of 99. **Action**: Correct the COUNT parameter value and reinvoke ISGECA.
08	*xxxx*00FE	**Meaning**: The RIBOUT length specified in parameter RIBOUTLN was not large enough to process the number of resource requests specified by parameter COUNT. **Action**: Correct the RIBOUTLN or COUNT parameter values and reinvoke ISGECA.
08	*xxxx*00FF	**Meaning**: ISGECA is an unsupported service on this system. **Action**: No suggested program action. The system needs a service or release level upgrade before ISGECA can be successfully invoked.

Examples

The following examples do not show, but presume, the existence of appropriate assembler continuation characters in column 72. The examples also presume an appropriate assembler storage declaration for each instance of a named symbol ISGECA parameter.

The first example depicts an invocation of ISGECA to collect waiter data for a specific sysplex system, whose 8-character system name is stored at program location MYSYSNAME:

```
        XR    2,2           Clear reg 2
        LHI   2,476         ISGECA parm list length into R2
        STORAGE OBTAIN,LENGTH=(2),ADDR=(3),SP=231,COND=NO
        GETWAIT  ISGECA REQUEST=WAITER,SCOPE=SYSTEM,
               SYSNAME=MYSYSNAME,RIBOUT=OUTAREA@,
               RIBOUTLN=MYOUTAREALEN,RIBOUTCT=MYRIBCT,
               COUNT=MYCOUNT,NOTINCL=NOTINCLAREA@,
               NOTINCCT=MYNOTCT,RETCODE=MYRETCODE,
               RSNCODE=MYRSNCODE,PLISTVER=MAX,
               MF=(E,(3))
```

For the above, subpool 231 storage is obtained and then passed through the MF= parameter for the ISGECA service routine parameter list.

Upon return from the service routine, the virtual storage area specified by OUTAREA@ contains the waiter report RIBs and RIBEs for up to MYCOUNT number of resources; while the virtual storage area specified by NOTINCLAREA@ contains the associated list of systems (with reasons) that are not included in the RIBOUT area report. The precise number of RIBs returned in the OUTAREA@ area is returned in the MYRIBCT program variable.

This second example depicts an invocation of ISGECA to collect blocker data for all the systems in the current sysplex:

```
       GETBLOCK ISGECA REQUEST=BLOCKER,SCOPE=SYSTEMS,
                    RIBOUT=OUTAREA@,RIBOUTLN=MYOUTAREALEN,
                    RIBOUTCT=MYRIBCT,COUNT=MYCOUNT,
                    NOTINCL=NOTINCLAREA@,NOTINCCT=MYNOTCT,
                    RETCODE=MYRETCODE,RSNCODE=MYRSNCODE,
                    PLISTVER=MAX,MF=S
```

Parameter usage and results for this example are analogous to the previous example. In this case, upon return from the ISGECA service routine, the virtual storage area specified by OUTAREA@ contains the blocker report RIBs and RIBEs. Note that the program itself must reside in common area subpool 231, because the ISGECA invocation is using the standard macro format.

Chapter 126. ISGENQ — Global resource serialization ENQ service

Description

Interface for Global Resource Serialization ENQ OBTAIN and RELEASE requests.

The GRS ENQ service routine is given control from the ISGENQ macro to:

- Obtain a single or multiple ENQs with or without associated device reserves.
- Change a single or multiple existing ENQs.
- Release a single or multiple ENQs.
- Test an obtain request.

This service is intended to replace ENQ, DEQ, and RESERVE.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. Any PSW key To use OWNINGTTOKEN, ENQMAX, or when the specified QNAME is one of the authorized QNAMEs, authorization must be one of the following: Supervisor state, PSW key 0-7, or APF authorized. **Note:** When an authorized caller issues an OBTAIN request with an unauthorized QNAME, if COND=YES, the request is granted, but a warning return code and the reason ISGENQRsn_UnprotectedQName are given. This is to warn that an unauthorized caller may block the ENQ, or even release the ENQ if running under the owning task. If COND=NO, authorized callers cannot obtain an ENQ on an unprotected resource. The authorized QNAMES are: ADRDFRAG ADRDSN ARCENQG BWODSN SYSCTLG SYSDSN SYSIEA01 SYSIEECT SYSIEFSD SYSIGGV1 SYSIGGV2 SYSPSWRD SYSVSAM SYSVTOC SYSZ*
Dispatchable unit mode:	Task

Environmental factor	Requirement
Cross memory mode:	Any PASN, any HASN, any SASN Note: The resulting ENQ is associated with the owning task in the home address space.
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in access register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The control parameters must be in the same key as the caller. The ECB specified must be in the caller's home address space or in common. The TCB of the owning task (the current task or specified by OWNINGTTOKEN) must be in the caller's home address space. If a captured UCB address is specified, the captured UCB must be in the caller's home address space.

Programming requirements

The caller must include the ISGYCON macro to get the return and reason codes.

The caller must include the ISGYENQ macro to get the mappings for the ISGYENQAA, ISGYENQRES, ISGYENQTOKEN, and ISGYENQRETURN tables.

See "Avoiding Interlock" in *z/OS MVS Programming: Assembler Services Guide* to ensure that you are following the required protocols to prevent the interlock.

Restrictions

The caller must not have functional recovery routines (FRRs).

This macro supports multiple versions. Some keywords are unique to certain versions. See the “,PLISTVER=IMPLIED_VERSION” on page 1215 parameter description.

Input register information

Before issuing the ISGENQ macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

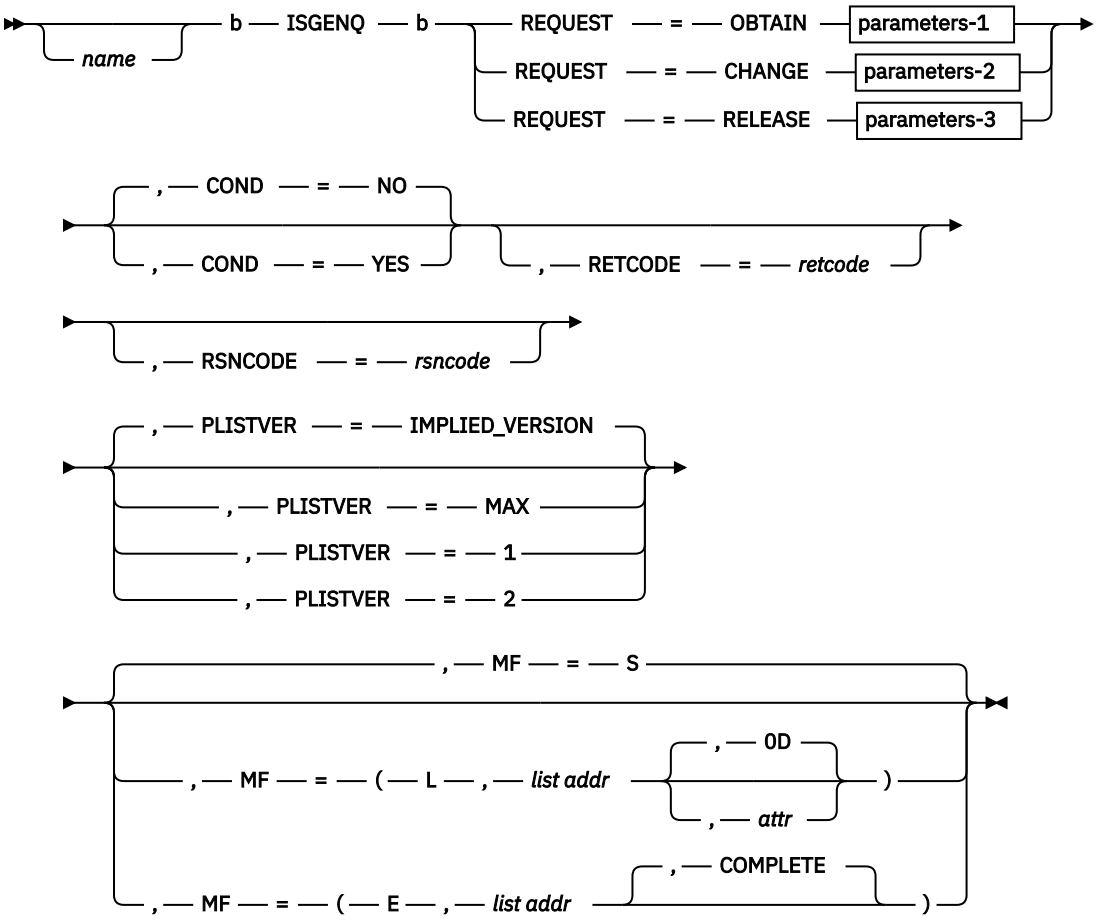
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

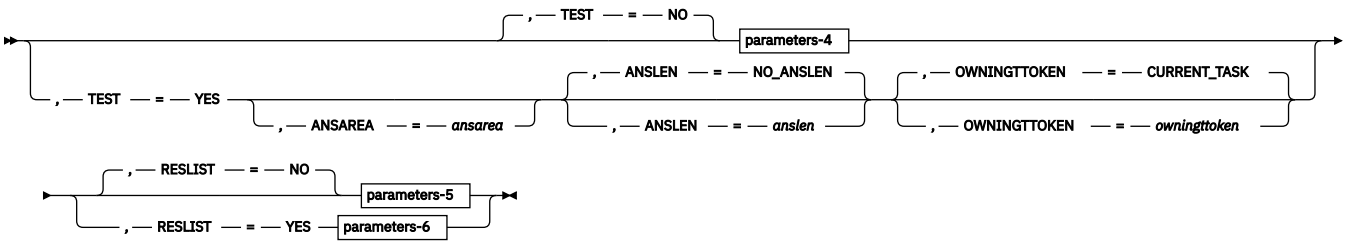
None.

Syntax

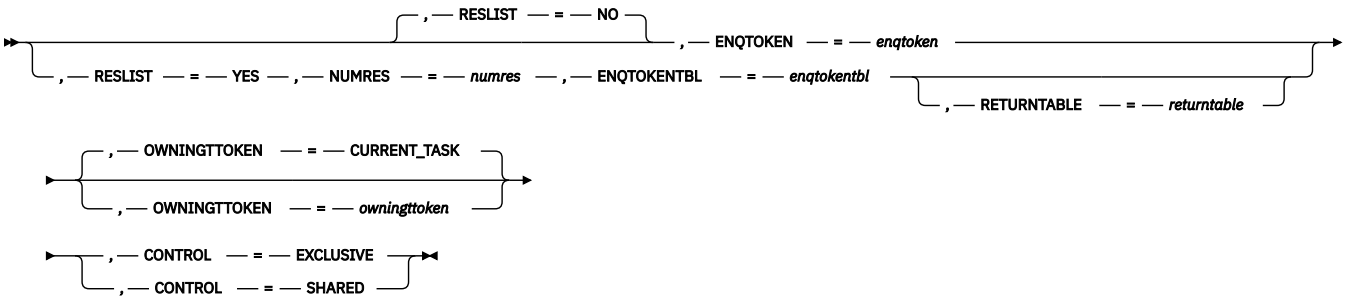
ISGENQ syntax



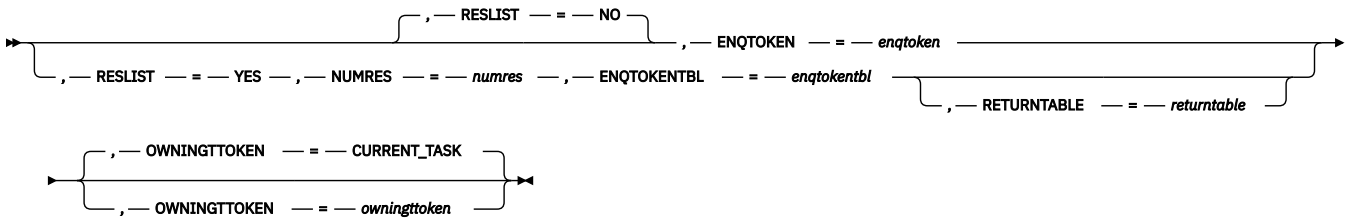
parameters-1



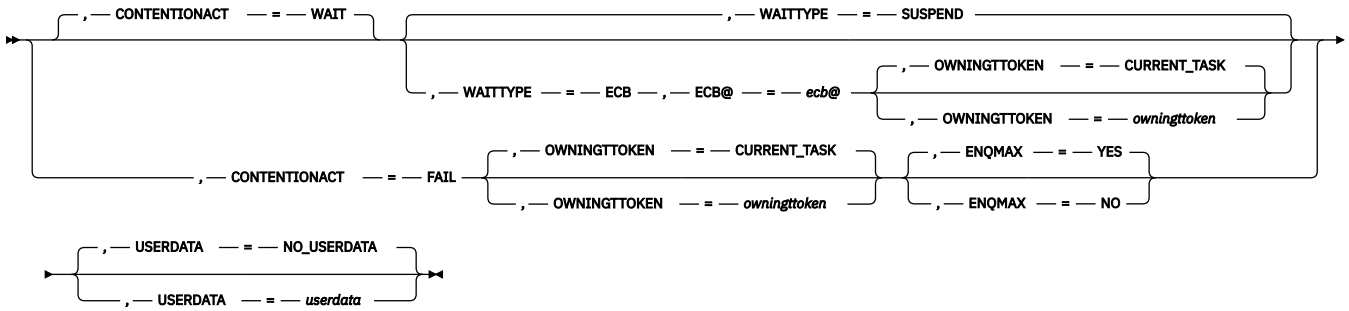
parameters-2



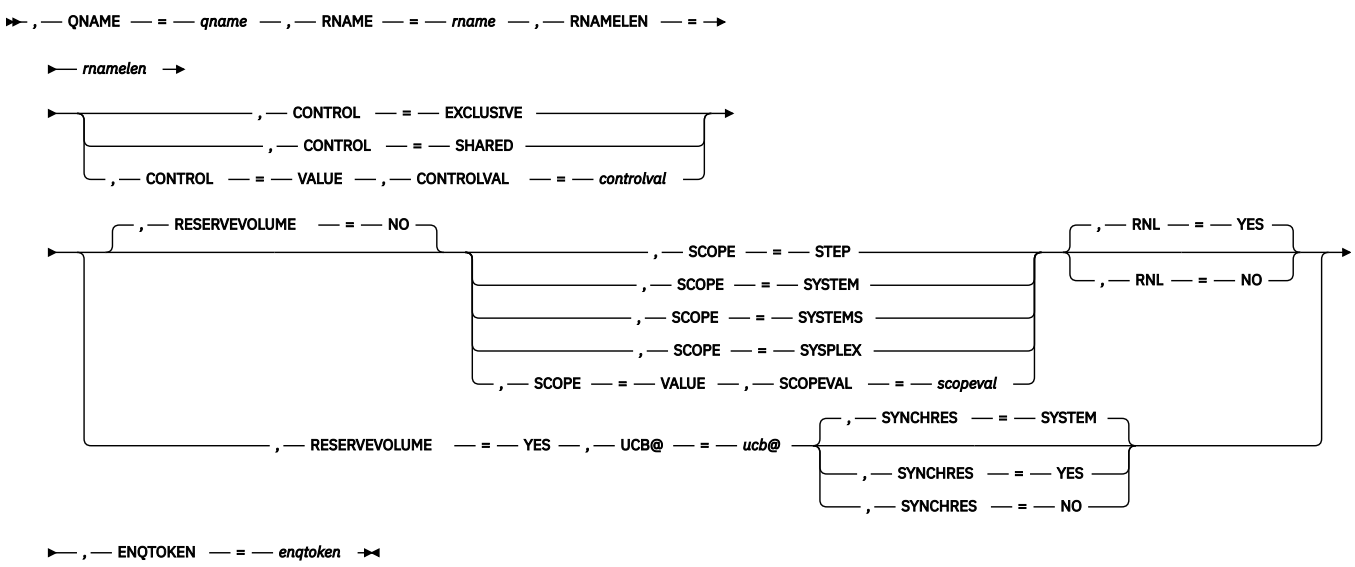
parameters-3



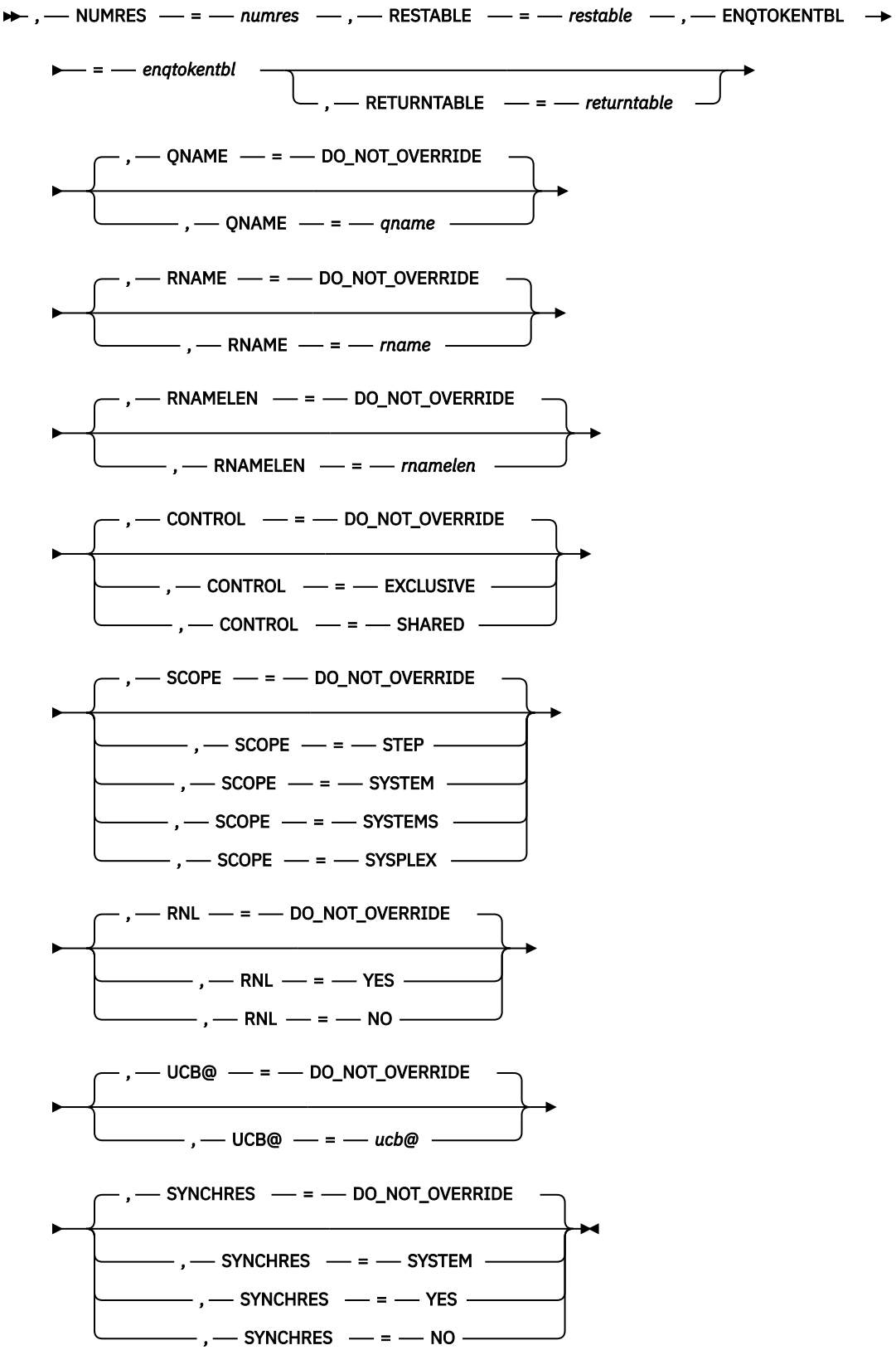
parameters-4



parameters-5



parameters-6



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the ISGENQ macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,ANSAREA=*ansarea*
When TEST=YES and REQUEST=OBTAIN are specified, an optional output parameter, which contains the returned information. The area is a list of records that are mapped by ISGYENQAA in the ISGYENQ macro. For RESLIST=YES, the records are in the same order as the requests in the RESTABLE. ANSLEN is required if ANSAREA is specified.

Note: The answer area is returned only when RC=0 or RC=4.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ENQMAX=YES
,ENQMAX=NO
When TEST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates whether ENQMAX checking should be done. This keyword tells global resource serialization whether a check is to be made to see if the limit for the number of concurrent resource requests has been exceeded. The default is ENQMAX=YES.

,ENQMAX=YES
Indicates ENQMAX checking should be done. IBM suggests that you use the default, ENQMAX=YES, to allow global resource serialization to perform this processing.

,ENQMAX=NO
Indicates that ENQMAX checking should not be used. Use ENQMAX=NO when you have a system-critical ENQ request that should be honored regardless of the concurrent number of resource requests made from the home address space.

Note: ENQMAX=NO can only be specified by an authorized requester and therefore can only override the maximum for authorized requesters.

See *z/OS MVS Planning: Global Resource Serialization* for more information.

,ANSLEN=*anslen*
,ANSLEN=NO_ANSLEN
When TEST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is the length of the answer area provided. The answer area should be large enough to hold an ISGYENQAA record and an RNAME for each request (specified by NUMRES, or one if RESLIST=NO). The maximum size area needed to contain one RNAME is 256 bytes. ANSAREA is required if ANSLEN is specified. The default is NO_ANSLEN.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,COND=NO
,COND=YES
An optional parameter that indicates how the request is handled for unsuccessful processing. The default is COND=NO.

,COND=NO
Indicates that if the request is not successful, then ISGENQ should ABEND the caller. COND=NO is mutually exclusive with RETCODE, RSNCODE, RETURNTABLE, WAITTYPE=ECB, and with TEST=YES.

,COND=YES
Indicates that ISGENQ should always return to the caller and indicate via return and reason codes whether the request was successful. If COND=YES is specified, RETCODE and RSNCODE (and RETURNTABLE, if RESLIST=YES) are required keywords.

Note: When COND=YES, ISGENQ tries to provide return and reason codes for the errors occurred during the process, though in some cases abends might be issued.

,CONTENTIONACT=WAIT
,CONTENTIONACT=FAIL
When TEST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates the action that should be taken if there is contention for the requested resource.

Note that a reserve request (where UCB@ is specified) that is not converted to only a global ENQ (Systems) will consist of an ENQ resource and a hardware reserve. For more information on reserve processing, see the description of the ",SYNCHRES=SYSTEM" on page 1220 keyword for more information on reserve processing. The default is CONTENTIONACT=WAIT.

,CONTENTIONACT=WAIT
Indicates that the caller waits until the ENQ resource is available and, if applicable, the synchronous reserve I/O (see SYNCHRES) is complete.

,CONTENTIONACT=FAIL
Indicates that if contention for the ENQ resource exists to cancel the ENQ obtain request and return to the caller.

Notes:

- See CONTENTIONACT=WAIT with ECB@ as a means of timing the overall request.
- For a reserve request (where UCB@ is specified), the ENQ resource is always obtained first. As such, CONTENTIONACT=FAIL indicates to cancel the entire request when there is contention on the ENQ resource. However, it does not apply to contention on the hardware reserve. See CONTENTIONACT=WAIT with WAITTYPE=ECB for information on how to manage or time hardware reserve contention.
- When SYNCRES is enabled, the request is subject to a delay for the reserve even if RET=USE or CONTENTIONACT=Fail is specified.

,CONTROL=EXCLUSIVE
,CONTROL=SHARED
,CONTROL=VALUE
When RESLIST=NO and REQUEST=OBTAIN are specified, a required parameter that is the control type of the ENQ to be obtained. If the resource is modified while under control of the task, the request must be for exclusive control. If the resource is not modified, the request should be for shared control.

,CONTROL=EXCLUSIVE
Indicates that the request is for exclusive control of the resource.

,CONTROL=SHARED
Indicates that the request is for shared control of the resource.

,CONTROL=VALUE
the user provides a value, through the CONTROLVAL keyword, indicating the requested control.

,CONTROL=DO_NOT_OVERRIDE
,CONTROL=EXCLUSIVE
,CONTROL=SHARED
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional parameter that is the type of control to be used for all resources specified in the resource table. This overrides any control specified in the resource table. If the resource is modified while under control of the task, the request must be for exclusive control. If the resource is not modified, the request should be for shared control. The default is CONTROL=DO_NOT_OVERRIDE.

,CONTROL=DO_NOT_OVERRIDE
Indicates that the control specified in the resource table should be used.

,CONTROL=EXCLUSIVE
Indicates that all requests are for exclusive control of the resources.

,CONTROL=SHARED
Indicates that all requests are for shared control of the resources.

,CONTROL=EXCLUSIVE
,CONTROL=SHARED
When RESLIST=NO and REQUEST=CHANGE are specified, control is an optional keyword input that is the control type to which the ENQ is to be changed. If the resource is modified under control of the task the request must be for exclusive control. If the resource is not modified, the request should be for shared control. When RESLIST=YES is specified, all resources in the list will be changed to the specified scope. The default is CONTROL=EXCLUSIVE.

,CONTROL=EXCLUSIVE
Indicates that the request is to change to exclusive control of the resource.

,CONTROL=SHARED
Indicates that the request is to change to shared control of the resource.

,CONTROLVAL=*controlval*
When CONTROL=VALUE, RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that contains a value indicating the desired control. The value provided must be equivalent to the constants provided in the ISGYENQ macro indicating the control. (See the ISGYENQ_kControl constants in the ISGYENQ macro for more information.)

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,ECB@=*ecb@*
When WAITTYPE=ECB, CONTENTIONACT=WAIT, TEST=NO and REQUEST=OBTAIN are specified, a required input parameter that contains the address of the ECB to be posted when the requested resource(s) is/are obtained.

The ECB must be in one of the following locations:

- the home address space of the caller.
- common space.
- for unauthorized requesters, in the same storage key as the requester.

When the ISGENQ service returns to the caller, the return and reason codes specify for each resource whether the task has been given control of the resource or needs to wait for the ECB to be posted.

When the ECB is posted, it contains a return/reason code pair. Bits 8-23 contain the low-order halfword of the reason code and bits 24-31 contain the low-order byte of the return code. For a RESLIST=NO request, the ECB contains the return and reason code for the request. For a RESLIST=YES request, the ECB contains an overall return code.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,ENQTOKEN=*enqtoken*
When RESLIST=NO and REQUEST=OBTAIN are specified, a required output parameter that is a token that uniquely identifies the ENQ. The ENQTOKEN is used on subsequent REQUEST=RELEASE or CHANGE invocations to release or change the ENQ request.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,ENQTOKEN=*enqtoken*
When RESLIST=NO and REQUEST=CHANGE are specified, a required input parameter that is an ENQ Token of the ENQ to be changed.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,ENQTOKEN=*enqtoken*
When RESLIST=NO and REQUEST=RELEASE are specified, a required input parameter that is an ENQ Token of the ENQ to be released.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,ENQTOKENTBL=*enqtokentbl*
When RESLIST=YES and REQUEST=OBTAIN are specified, a required output parameter that is a table of ENQ tokens. Mapped by ISGYENQToken in the ISGYENQ macro. To easily release any ENQs obtained by a REQUEST=OBTAIN use the same ENQToken table as input to a REQUEST=RELEASE.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ENQTOKENTBL=*enqtokentbl*
When RESLIST=YES and REQUEST=CHANGE are specified, a required input parameter that is a table of ENQ Tokens. Mapped by ISGYENQToken in the ISGYENQ macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ENQTOKENTBL=*enqtokentbl*
When RESLIST=YES and REQUEST=RELEASE are specified, a required input parameter that is a table of ENQ Tokens. Mapped by ISGYENQToken in the ISGYENQ macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NUMRES=*numres*
When RESLIST=YES and REQUEST=OBTAIN are specified, a required input parameter that is the number of resource entries in the resource table. The specified value can be in the range of 1 to 2?6-1 (65535).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,NUMRES=*numres*
When RESLIST=YES and REQUEST=CHANGE are specified, a required input parameter that is the number of ENQ tokens in the ENQ token table. The specified value can be in the range of 1 to 2?6-1 (65535).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,NUMRES=*numres*
When RESLIST=YES and REQUEST=RELEASE are specified, a required input parameter that is the number of ENQ tokens in the ENQ Token Table. The specified value can be in the range of 1 to 2?6-1 (65535).

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,OWNINGTTOKEN=*owningttoken*
,OWNINGTTOKEN=CURRENT_TASK
When WAITTYPE=ECB, CONTENTIONACT=WAIT, TEST=NO and REQUEST=OBTAIN are specified, an optional input parameter that is the task token (TToken) of the task on whose behalf the ENQ is to be obtained. The TToken must specify a task in the caller's home address space.

Note: Mutually exclusive with RESERVEVOLUME=YES. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,OWNINGTTOKEN=*owningttoken*
,OWNINGTTOKEN=CURRENT_TASK
When CONTENTIONACT=FAIL, TEST=NO and REQUEST=OBTAIN are specified, an optional input parameter that is the task token (TToken) of the task on whose behalf the ENQ is to be obtained. The TToken must specify a task in the caller's home address space.

Note: Mutually exclusive with RESERVEVOLUME=YES. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,OWNINGTTOKEN=*owningttoken*
,OWNINGTTOKEN=CURRENT_TASK
When TEST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is the task token (TToken) of the task on whose behalf the test request is to be performed. The TToken must specify a task in the caller's home address space.

Note: Mutually exclusive with RESERVEVOLUME=YES. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,OWNINGTTOKEN=*owningttoken*
,OWNINGTTOKEN=CURRENT_TASK
When REQUEST=CHANGE is specified, an optional input parameter that is the task token (TToken) of the task that owns the ENQ that is to be changed. The TToken must specify a task in the caller's home address space. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,OWNINGTTOKEN=*owningttoken*
,OWNINGTTOKEN=CURRENT_TASK
When REQUEST=RELEASE is specified, an optional input parameter that is the task token (TToken) of the task that owns the ENQs that are to be released. The TToken must specify a task in the caller's home address space. The default is CURRENT_TASK.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
,PLISTVER=2
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, which supports all parameters except those specifically referenced in higher versions.
- **2**, which supports both the following parameters and those from version 1: USERDATA

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1, or 2

,QNAME=*qname*
When RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that is the QNAME of the resource. The QNAME can contain any character from X'00' to X'FF'. However, a unique readable value that identifies the functional area or a high level of what is being serialized is preferred. Every program issuing a request for a serially reusable resource must use the same QNAME, RNAME, and Scope to represent the resource. Some names, such as those beginning with certain letter combinations (SYSZ for example), are used to protect system resources by requiring that the issuing program be in supervisor state, or system key, or APF-authorized. Authorized programs must use a restricted QNAME (as described under Minimum authorization in the Environment section for this service) to prevent interference from unauthorized programs.

For a list of QNAME (also known as major name) and RNAME (also known as minor name) ENQ or DEQ names and the resources that issue the ENQ or DEQ, see *z/OS MVS Diagnosis: Reference*.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,QNAME=*qname*
,QNAME=DO_NOT_OVERRIDE
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is a common QNAME to be used for all resources in the resource table. This overrides any QNAMEs specified in the resource table. The QNAME can contain any character from X'00' to X'FF'. However, a unique readable value that identifies the functional area or a high level of what is being serialized is preferred. Every program issuing a request for a serially reusable resource must use the same QNAME, RNAME, and Scope to represent the resource. Some names, such as those beginning with certain letter combinations (SYSZ for example), are used to protect system resources by requiring that the issuing program be in supervisor state, or system key, or APF-authorized. Authorized programs must use a restricted QNAME (as described under Minimum authorization in the Environment section for this service) to prevent interference from unauthorized programs.

For a list of QNAME (also known as major name) and RNAME (also known as minor name) ENQ or DEQ names and the resources that issue the ENQ or DEQ, see *z/OS MVS Diagnosis: Reference*.

The default is DO_NOT_OVERRIDE.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

REQUEST=OBTAIN
REQUEST=CHANGE
REQUEST=RELEASE
A required parameter that indicates the type of ISGENQ request.

REQUEST=OBTAIN
Indicates a request to obtain an ENQ for a resource.

REQUEST=CHANGE
Indicates a request to change the status an ENQ from shared to exclusive control.

REQUEST=RELEASE
Indicates a request to release (dequeue) the ENQ for a resource.

,RESERVEVOLUME=NO
,RESERVEVOLUME=YES
When RESLIST=NO and REQUEST=OBTAIN are specified, an optional parameter. The default is RESERVEVOLUME=NO.

,RESERVEVOLUME=NO
Indicates to issue a normal ENQ obtain and not a reserve.

,RESERVEVOLUME=YES
Indicates that after the ENQ resource is obtained that a reserve for the given device (shared DASD) is to be issued.

Note: RESERVEVOLUME=YES is mutually exclusive with OWNINGTTOKEN.

,RESLIST=NO
,RESLIST=YES
When REQUEST=OBTAIN is specified, an optional parameter, The default is RESLIST=NO.

,RESLIST=NO
Indicates to obtain an ENQ for a single resource.

,RESLIST=YES
Indicates to obtain ENQs for multiple resources specified in a resource table. Specifying multiple requests in a list ensures that they are processed atomically with respect to other ISGENQ requests. However, the order in which the requests are processed is unpredictable. Each request is treated as a separate request, and if COND=YES is specified, then the return code for each request should be checked.

Note: An easy way to release a list of ENQs is to use the output ENQTOKEN table from the OBTAIN request as input to a RELEASE request.

,RESLIST=NO
,RESLIST=YES
When REQUEST=CHANGE is specified, an optional parameter, The default is RESLIST=NO.

,RESLIST=NO
Indicates to change the control of a single ENQ.

,RESLIST=YES
Indicates to change the control for multiple ENQs.

,RESLIST=NO
,RESLIST=YES
When REQUEST=RELEASE is specified, an optional parameter, The default is RESLIST=NO.

,RESLIST=NO
Indicates to single ENQ RELEASE request.

,RESLIST=YES
Indicates to change the disposition for multiple ENQs.

Note: A easy way to release a list of ENQs is to use the output ENQTOKEN table from the OBTAIN request as input to a RELEASE request.

,RESTABLE=*restable*
When RESLIST=YES and REQUEST=OBTAIN are specified, a required input parameter that is a table specifying multiple ENQ requests. The resource table is mapped by ISGYENQRes in the ISGYENQ macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RETURNTABLE=*returntable*
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional output parameter that is a table that contains the return and reason codes. Mapped by ISGYENQReturn in the ISGYENQ macro. The return table is only valid when ISGENQRsn_NonZeroReturnCodes is returned in the RSNCODE. Mutually exclusive with COND=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RETURNTABLE=*returntable*
When RESLIST=YES and REQUEST=CHANGE are specified, an optional output parameter that is a table that contains the return and reason codes. Mapped by ISGYENQReturn in the ISGYENQ macro. The return table is only valid when ISGENQRsn_NonZeroReturnCodes is returned in the RSNCODE. Mutually exclusive with COND=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RETURNTABLE=*returntable*
When RESLIST=YES and REQUEST=RELEASE are specified, an optional output parameter that is a table that contains the return and reason codes. Mapped by ISGYENQReturn in the ISGYENQ macro. The return table is only valid when ISGENQRsn_NonZeroReturnCodes is returned in the RSNCODE. Mutually exclusive with COND=NO.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RNAME=*rname*
When RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that is the RNAME for the resource. The RNAME must be from 1 to 255 bytes long, and can contain any hexadecimal character from X'00' to X'FF'.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RNAME=*rname*
,RNAME=DO_NOT_OVERRIDE
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is the common RNAME to be used for all resources in the resource table. This overrides any RNAMEs specified in the resource table. The RNAME must be from 1 to 255 bytes long, and can contain any hexadecimal character from X'00' to X'FF'. The default is DO_NOT_OVERRIDE.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RNAMELEN=*rnamelen*
When RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that is the length of the given RNAME. The specified length can be in the range of 1 to 255.

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,RNAMELEN=*rnamelen*
,RNAMELEN=DO_NOT_OVERRIDE
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional input parameter that is a common length to be used for all RNAMEs in the resource table, or if a common RNAME is specified, it is the length of the common RNAME. The specified length can be in the range of 1 to 255. This overrides any RNAMEs lengths specified in the resource table. The default is DO_NOT_OVERRIDE.

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,RNL=YES
,RNL=NO
When RESERVEVOLUME=NO, RESLIST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates whether the scope can be changed by global resource serialization resource name list (RNL) processing, installation exits, or alternate serialization products. The default is RNL=YES.

,RNL=YES
Indicates that global resource serialization RNL processing should be used, which can cause the scope of a resource to change. In general, IBM recommends that you use the default, RNL=YES, to allow global resource serialization to perform RNL processing and allow installation exits and alternate serialization products to alter, extend, or restrict the scope and other attributes, as needed.

,RNL=NO
Indicates that global resource serialization RNL processing should not be used. The scope of the resource is not changed by the RNLs, any installation exits, or alternate serialization product. Use RNL=NO when you are sure that you want the request to be processed only by GRS using only the specified scope or when an alternate serialization product or installation exit should be prevented from altering the scope or other attributes, extending the scope beyond the GRS serialization complex, or restricting the scope to systems other than all of those in the GRS complex.

,RNL=DO_NOT_OVERRIDE
,RNL=YES
,RNL=NO
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional parameter that indicates whether the scope can be changed by global resource serialization resource name list (RNL) processing or the dynamic exits. This overrides any RNL processing specified in the resource table. The default is RNL=DO_NOT_OVERRIDE.

,RNL=DO_NOT_OVERRIDE
Indicates that the RNL specifications in the resource table should be used.

,RNL=YES
Indicates that global resource serialization RNL processing should be used, which can cause the scope of a resource to change. IBM suggests that you use the default, RNL=YES, to allow global resource serialization to perform RNL processing.

,RNL=NO
Indicates that global resource serialization RNL processing should not be used. The scope of the resource cannot be changed by the RNLs or any dynamic exits. Use RNL=NO when you are sure that you want the request to be processed only by global resource serialization using only the specified scope. When RNL=NO is specified, the ENQ request is ignored by alternative serialization products.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
,SCOPE=SYSPLEX
,SCOPE=VALUE
When RESERVEVOLUME=NO, RESLIST=NO and REQUEST=OBTAIN are specified, a required parameter that is the scope of the resource.

,SCOPE=STEP
Indicates that the resource is serialized only within an address space. If STEP is specified, a request for the same QNAME and RNAME from a program in another address space denotes a different resource.

,SCOPE=SYSTEM
Indicates that the resource is serialized across all address spaces in a system.

,SCOPE=SYSTEMS
Indicates that the resource is serialized across all systems in a GRS Star or GRS Ring complex.

,SCOPE=SYSPLEX
Indicates that the resource is serialized across all systems in a GRS Star sysplex or GRS ring. (Same as scope SYSTEMS.)

,SCOPE=VALUE
the user provides a value, through the SCOPEVAL keyword, indicating the requested scope.

,SCOPE=DO_NOT_OVERRIDE
,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
,SCOPE=SYSPLEX
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional parameter that is the scope to be used for all resources in the resource table. This overrides any scopes specified in the resource table. The default is SCOPE=DO_NOT_OVERRIDE.

,SCOPE=DO_NOT_OVERRIDE
Indicates that the scope specified in the resource table should be used.

,SCOPE=STEP
Indicates that the resource is serialized only within an address space. If STEP is specified, a request for the same QNAME and RNAME from a program in another address space denotes a different resource.

,SCOPE=SYSTEM
Indicates that the resource is serialized across all address spaces in a system.

,SCOPE=SYSTEMS
Indicates that the resource is serialized across all systems in a GRS Star or GRS Ring complex.

,SCOPE=SYSPLEX
Indicates that the resource is serialized across all systems in a GRS Star sysplex or GRS ring. (Same as scope SYSTEMS.)

,SCOPEVAL=*scopeval*
When SCOPE=VALUE, RESERVEVOLUME=NO, RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that contains a value indicating the desired scope. The value provided must be equivalent to the constants provided in the ISGYENQ macro indicating the scope. (See the ISGYENQ_ constants in the ISGYENQ macro for more information.)

To code: Specify the RS-type address, or address in register (2)-(12), of an one-byte field.

,SYNCHRES=SYSTEM
,SYNCHRES=YES
,SYNCHRES=NO
When RESERVEVOLUME=YES, RESLIST=NO and REQUEST=OBTAIN are specified, an optional parameter that specifies whether the request should issue a synchronous reserve. A synchronous reserve immediately reserves the volume instead of waiting for the first use.

Note that an RC=4 (ISGENQRc_Warn), RSC=0403 (ISGENQRsn_ECBWillBePosted) is presented for CONTENTIONACT=WAIT, WAITTYPE=ECB, reserve requests (where UCB@ is specified) when there is contention on the ENQ resource or there was no contention on the resource, and the reserve I/O was done synchronously. The default is SYNCHRES=SYSTEM.

,SYNCHRES=SYSTEM
Indicates that the installation system default SYNCHRES setting should be used.

,SYNCHRES=YES
Indicates to issue a synchronous reserve. In cases where the hardware reserve is performed (it was not converted to a Global/Systems ENQ), the caller is ensured that the reserve I/O is complete when the ISGENQ request has successfully completed.

,SYNCHRES=NO
Indicates that a synchronous reserve should be avoided when possible. Some devices require that the reserve must be done synchronously regardless of this setting. If the reserve I/O is not done

synchronously, the reservce is done when the first I/O is done to the device after the reserve request is issued. For more information, see *z/OS MVS Planning: Global Resource Serialization*.

,SYNCHRES=DO_NOT_OVERRIDE
,SYNCHRES=SYSTEM
,SYNCHRES=YES
,SYNCHRES=NO

When RESLIST=YES and REQUEST=OBTAIN are specified, an optional parameter that specifies whether all requests specified in the resource table should issue a synchronous reserve. This overrides any SYNCHRES specified in the resource table. A synchronous reserve immediately reserves the volume instead of waiting for the first use. The default is SYNCHRES=DO_NOT_OVERRIDE.

,SYNCHRES=DO_NOT_OVERRIDE
Indicates that the SYNCHRES specified in the resource table should be used.

,SYNCHRES=SYSTEM
Indicates that the system default setting should be used.

,SYNCHRES=YES
Indicates to issue a synchronous reserve. In cases where the the hardware reserve is performed (it was not converted to a Global/Systems ENQ), the caller is ensured that the reserve I/O is complete when the request has successfully completed.

,SYNCHRES=NO
Indicates that a synchronous reserve should be avoided when possible. Some devices require that the reserve must be done synchronously regardless of this setting. If the reserve I/O is not done synchronously, the reserve is done when the first I/O is done to the device after the reserve request is issued. See *z/OS MVS Planning: Global Resource Serialization* for more information.

,TEST=NO
,TEST=YES

When REQUEST=OBTAIN is specified, an optional parameter. The default is TEST=NO.

,TEST=NO
Indicates that this is not a test request. The ENQ must be obtained.

,TEST=YES
Indicates that this is a test request. The ENQ must not be obtained. This parameter setting can be used to obtain information about how the given obtain request is processed and how a resource is currently held by the current task or a task specified by OWNINGTTOKEN.

Mutually exclusive with COND=NO.

For existing requests from the same task, which match the specified resource, the ENQToken of that request is returned.

See ISGQUERY SEARCH=BY_ENQTOKEN for information about outstanding ENQ requests.

The following return and reason codes can be used to determine if the resource is available and how it might be held by the OWNINGTTOKEN task:

- ISGENQRc_ok
- ISGENQRsn_NotImmediatelyAvailable
- ISGENQRsn_TaskOwnsExclusive
- ISGENQRsn_TaskOwnsShared
- ISGENQRsn_TaskWaiting

,UCB@=*ucb@*

When RESERVEVOLUME=YES, RESLIST=NO and REQUEST=OBTAIN are specified, a required input parameter that contains the address of the UCB for the device to be reserved. For unauthorized callers, the UCB must be allocated to the job step before ISGENQ RESERVEVOLUME(YES) is issued.

Note: Authorized callers do not need to allocate the UCB to the job step before invoking ISGENQ, but the caller must serialize the UCB against dynamic I/O reconfiguration requests. The caller can accomplish this serialization by allocating or pinning the UCB. Such serialization ensures that a

dynamic I/O reconfiguration request does not delete or reuse the UCB before the ISGENQ macro uses the address.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,UCB@=*ucb@*
,UCB@=DO_NOT_OVERRIDE
When RESLIST=YES and REQUEST=OBTAIN are specified, an optional input parameter that contains the address of the UCB@ for the device to be reserved for all resources in the resource table. This overrides any UCB addresses specified in the resource table. The default is DO_NOT_OVERRIDE.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

,USERDATA=*userdata*
,USERDATA=NO_USERDATA
When TEST=NO and REQUEST=OBTAIN are specified, an optional input parameter that contains the userdata to be associated with this request. For information about using USERDATA as a filter, or making ISGQUERY return USERDATA for requests, see Chapter 139, “ISGQUERY — Global resource serialization query service,” on page 1305.

Note that GRS has no interests in the contents of the USERDATA. Unlike the QNAME, RNAME, and SCOPE parameters, USERDATA has no meaning in the definition of the logically serialized resource identity. For example, exclusive requests with different user data and the same QNAME, RNAME, and SCOPE contend with each other.

This request requires a version 2 parameter list. The default is NO_USERDATA.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,WAITTYPE=SUSPEND
,WAITTYPE=ECB
When CONTENTIONACT=WAIT, TEST=NO and REQUEST=OBTAIN are specified, an optional parameter that indicates the method by which the caller waits. The default is WAITTYPE=SUSPEND.

,WAITTYPE=SUSPEND
Indicates that the current task is suspended until the entire request is completed.

,WAITTYPE=ECB
Indicates that if contention for the ENQ resource exists or the device reserve is done synchronously (see “,SYNCHRES=SYSTEM” on page 1220), return to the caller, and post the ECB when the request is complete.

Mutually exclusive with COND=NO.

WAITTYPE=ECB in combination with setting a timer with ECB can be used to control the amount of time that you are willing to wait for either ENQ contention or a synchronous reserve to complete. If the request does not complete before the time expires you can do the following actions.

- You can use the ISGECA and ISGQUERY services to interrogate the overall state of the request and associated resource.
- You can back out of the request using an ISGENQ REQUEST=RELEASE request."

ABEND codes

For REQUEST=OBTAIN and REQUEST=CHANGE requests the caller might encounter abend codes X'138', X'238', X'338', X'438', X'538', X'738', X'838', X'938'.

For REQUEST=RELEASE requests the caller might encounter abend codes X'130', X'230', X'330', X'430', X'530', X'730', X'830', X'930'.

For explanations and responses for these codes, see *z/OS MVS System Codes*.

Note that the ABEND reason codes correspond to the same reason codes listed in Table 112 on page 1223.

Return and reason codes

When the ISGENQ macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro ISGYCON provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support the *xxxx* value, where *xxxx* represent 4 hex digits. Note that when the *xxxx* value is 'E0F2' hexadecimal, it indicates a reason-code set by the ISGNQXITBATCH or ISGNQXITBATCHCND exits.

Table 112. Return and Reason Codes for the ISGENQ Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
00	—	**Equate Symbol**: ISGENQRc_OK **Meaning**: ISGENQ request successful. Depending on the type of request, the ENQ is successfully obtained, changed to exclusive, or released. If RESLIST=YES is specified, all ENQ obtain, change, and release requests are successful. For REQUEST=OBTAIN, TEST=YES, the resource is immediately available. **Action**: None required.
04	—	**Equate Symbol**: ISGENQRc_Warn **Meaning**: Warning **Action**: Refer to action under the individual reason code.
04	xxxx0401	**Equate Symbol**: ISGENQRsn_NonZeroReturnCodes **Meaning**: A non-zero return code was issued for one or more entries in a RESLIST=YES request. The return table has the return and reason codes for each of the requests in the list. **Action**: See the return and reason codes returned in the RETURNTABLE.
04	xxxx0402	**Equate Symbol**: ISGENQRsn_RequestNotProcessed **Meaning**: For RESLIST=YES requests. One of the other requests in the RESTABLE failed such that this request was prevented from being processed. Note that requests in a RESTABLE are not necessarily processed in the order they appear in the RESTABLE. Note: This reason code returned only in the RETURNTABLE, not through the RSNCODE keyword. **Action**: Check the return and reason codes for all other requests in the RETURNTABLE to identify the problem.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
04	xxxx0403	**Equate Symbol**: ISGENQRsn_ECBWillBePosted **Meaning**: For REQUEST=OBTAIN CONTENTIONACT=WAIT WAITTYPE=ECB, the OBTAIN request was successful, but the ENQ resource was not immediately available or the reserve I/O needed to be done synchronously (SYNCHRES). The ECB is posted when all requested resources are owned by the specified task, or when an error has occurred. The ENQToken for the request has been returned. **Action**: Wait on the ECB and check the return code in the ECB before using the requested resources.
04	xxxx0404	**Equate Symbol**: ISGENQRsn_NotImmediatelyAvailable **Meaning**: The ENQ of the resource was not immediately available. For REQUEST=OBTAIN CONTENTIONACT=FAIL, the requested resource is not obtained. For REQUEST=OBTAIN TEST=YES, the holder is a task other than OWNINGTTOKEN. **Action**: No action required.
04	xxxx0405	**Equate Symbol**: ISGENQRsn_TaskOwnsExclusive **Meaning**: For REQUEST=OBTAIN, including TEST=YES, the given task specified by OWNINGTTOKEN already owns the specified resource exclusively. The ENQToken for the owning request has been returned. **Action**: No action required.
04	xxxx0406	**Equate Symbol**: ISGENQRsn_TaskOwnsShared **Meaning**: For a REQUEST=OBTAIN, including TEST=YES, the given task specified by OWNINGTTOKEN already owns the specified resource shared. The ENQToken for the owning request has been returned. **Action**: No action required.
04	xxxx0407	**Equate Symbol**: ISGENQRsn_TaskWaiting **Meaning**: For a REQUEST=OBTAIN, including TEST=YES, the given task specified by OWNINGTTOKEN is already waiting for control of the specified resource. The ENQToken for the waiting request has been returned. **Action**: No action required.
04	xxxx0409	**Equate Symbol**: ISGENQRsn_OtherSharedOwners **Meaning**: For REQUEST=CHANGE. The control cannot be changed to exclusive. There are other shared owners of the resource. **Action**: No action required.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
04	xxxx040A	**Equate Symbol**: ISGENQRsn_TaskDoesNotOwn **Meaning**: For REQUEST=CHANGE. The control cannot be changed to exclusive. The task does not yet own the resource. **Action**: No action required.
04	xxxx040B	**Equate Symbol**: ISGENQRsn_TaskSuspendedForResource **Meaning**: For REQUEST=RELEASE. The task that requested the ENQ obtain has not yet been assigned control of the resource The task continues waiting and the resource is not released. (This reason code might result in an exit routine, which received control because of an interruption, issued a RELEASE request on behalf of the task.) **Action**: Correct the program so that the ISGENQ RELEASE request is issued only after the ISGENQ OBTAIN request has returned to the task. If possible, avoid issuing the RELEASE request in the exit routine.
04	xxxx040D	**Equate Symbol**: ISGENQRsn_UnprotectedQName **Meaning**: For REQUEST=OBTAIN. An authorized caller requested an ENQ with an unauthorized QNAME. For TEST=NO,COND=YES, the OBTAIN request completed successfully, an unauthorized caller under the same owning task might release the ENQ. The ENQToken has been returned. For TEST=NO, COND=NO, the requester was abended with a X'438' abend. The request might not have completed successfully For TEST=YES requests, the resource is currently available. **Action**: No action required. If the ENQ needs to be protected from unauthorized RELEASE requests or from unauthorized callers obtaining an ENQ to block this request, specifiy one of the authorized QNAMEs for the resource.
04	xxxx040E	**Equate Symbol**: ISGENQRsn_UnprotectedExitQNAME **Meaning**: For REQUEST=OBTAIN. An authorized caller requested an ENQ with a QNAME that a dynamic exit changed to an unauthorized QNAME. For TEST=NO, the OBTAIN request completed successfully, an unauthorized caller under the same owning task might release the ENQ. The ENQToken has been returned. For TEST=YES requests, the resource is currently available but the QNAME was changed by a dynamic exit to an unprotected QNAME. **Action**: No action required. Contact the system programmer, if the ENQ needs to be protected from unauthorized RELEASE requests or from unauthorized callers obtaining an ENQ to block this request. The system programmer should check the ISGNQXIT installation exits to ensure that they are not coded to specify an unauthorized QNAME for authorized requests.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
04	xxxx040F	**Equate Symbol**: ISGENQRsn_ECBAtleastOneRequestFailed **Meaning**: For REQUEST=OBTAIN RESLIST=Yes with ECB@, at least one request failed to be processed. Some requests might have been processed unsuccessfully. The system might not backout any successfully processed requests. Note: This reason code is returned in a posted ECB, not through the RSNCODE or RETURNTABLE keywords. **Action**: The user should issue an ISGQUERY on the ENQTOKENs to see if they were obtained and take appropriate action. Alternately, the user can release all the ENQs with a ISGENQ REQUEST=RELEASE with ENQTOKENTBL and reissue the ISGENQ OBTAIN request.
08	—	**Equate Symbol**: ISGENQRc_ParmError **Meaning**: ISGENQ request specified parameters in error. **Action**: Refer to action under the individual reason code.
08	xxxx0801	**Equate Symbol**: ISGENQRsn_BadPlistAddress **Meaning**: Unable to access parameter list. **Action**: Check that the entire parameter list is addressable. If in AR-mode, check that the ALET of the parameter list is correct. Note that if this macro is issued in AR-mode, SYSSTATE ASCENV=AR must be issued before this macro. Ensure that the storage is in the same key as the caller.
08	xxxx0802	**Equate Symbol**: ISGENQRsn_BadPlistALET **Meaning**: Bad parameter list ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the parameter list is valid. Its access register may not have been set up properly.
08	xxxx0803	**Equate Symbol**: ISGENQRsn_BadPlistVersion **Meaning**: Bad parameter list version number. The service level of GRS on which the caller is running does not support this version of the ISGENQ service, or the ISGENQ parameter list version is lower than the minimum required for parameters that were specified. **Action**: Check for possible storage overlay of the parameter list. Retry the request with the correct version number. Verify that your program was assembled with the correct macro library for the release of MVS on which your program is running.
08	xxxx0804	**Equate Symbol**: ISGENQRsn_ReservedFieldNotNull **Meaning**: A reserved field in the parameter list is non-zero. **Action**: Check for possible storage overlay of the parameter list.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0805	**Equate Symbol**: ISGENQRsn_MutuallyExclusive **Meaning**: Mutually exclusive keywords were specified. **Action**: Check for a possible storage overlay of the parameter list.
08	xxxx0806	**Equate Symbol**: ISGENQRsn_BadRequest **Meaning**: Bad REQUEST parameter. **Action**: IBM suggests that the ISGENQ macro is used when invoking the ISGENQ service.
08	xxxx0807	**Equate Symbol**: ISGENQRsn_BadContentionAct **Meaning**: Bad CONTENTIONACT parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0808	**Equate Symbol**: ISGENQRsn_BadOwningTToken **Meaning**: The specified TToken does not represent a valid task. **Action**: Ensure that the task token (TToken) represents a valid task.
08	xxxx0809	**Equate Symbol**: ISGENQRsn_BadAnsAreaAddress **Meaning**: Unable to access the answer area. **Action**: Ensure that the entire answer area is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the specified answer area length is correct. Ensure that the storage is in the same key as the caller.
08	xxxx080A	**Equate Symbol**: ISGENQRsn_BadAnsAreaALET **Meaning**: Bad answer area ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the answer area is valid. Its access register may not have been set up properly.
08	xxxx080B	**Equate Symbol**: ISGENQRsn_AnsLenTooSmall **Meaning**: The specified answer area length was too small to return the requested information. **Action**: Invoke ISGENQ again with a larger answer area. The answer area length needed is dependent on the number of resource requests specified in NUMRES.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
08	xxxx080C	**Equate Symbol**: ISGENQRsn_BadRNameAddress **Meaning**: Unable to access the RNAME. **Action**: Ensure that the entire RNAME is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the specified RNAME length is correct. Ensure that the storage is in the same key as the caller.
08	xxxx080D	**Equate Symbol**: ISGENQRsn_BadRnameALET **Meaning**: Bad RNAME ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the RNAME is valid. Its access register may not have been set up properly.
08	xxxx080E	**Equate Symbol**: ISGENQRsn_BadRNameLen **Meaning**: The RNAME length specified is not valid. **Action**: Ensure the RNAME length field contains a number in the range of 1-255.
08	xxxx080F	**Equate Symbol**: ISGENQRsn_BadScope **Meaning**: Bad SCOPE keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0810	**Equate Symbol**: ISGENQRsn_BadUCB@ **Meaning**: The storage specified by the UCB@ keyword does not map to a valid UCB. **Action**: Ensure that the UCB@ points to a valid UCB.
08	xxxx0811	**Equate Symbol**: ISGENQRsn_BadCond **Meaning**: Bad COND keyword parameter. **Action**: IBM suggests that the ISGENQ macro is used when invoking the ISGENQ service.
08	xxxx0812	**Equate Symbol**: ISGENQRsn_BadSynchRes **Meaning**: Bad SYNCHRES keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0813	**Equate Symbol**: ISGENQRsn_BadENQTokenAddress **Meaning**: Unable to access the ENQToken. **Action**: Ensure that the entire ENQToken is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller. Note: The ISGENQ request might not have completed.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0814	**Equate Symbol**: ISGENQRsn_BadENQTokenALET **Meaning**: Bad ENQToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the ENQToken is valid. Its access register may not have been set up properly. Note: The ISGENQ request might not have completed.
08	xxxx0815	**Equate Symbol**: ISGENQRsn_BadENQToken **Meaning**: For REQUEST=RELEASE or REQUEST=CHANGE, the specified ENQToken does not represent an ENQ for the given task (current task or specified by OWNINGTTOKEN). **Action**: Ensure that the specified ENQToken is from a previous request for the given task, that has not been subsequently released.
08	xxxx0816	**Equate Symbol**: ISGENQRsn_BadNumRes **Meaning**: The NUMRES specified is not valid. **Action**: Ensure the NUMRES field contains a number in the range of 1-65535 (2?6-1)
08	xxxx0817	**Equate Symbol**: ISGENQRsn_BadResTableAddress **Meaning**: Unable to access the resource table. **Action**: Ensure that the entire resource table is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the resource table length is correct. Ensure that the storage is in the same key as the caller.
08	xxxx0818	**Equate Symbol**: ISGENQRsn_BadResTableALET **Meaning**: Bad resource table ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the resource table is valid. Its access register may not have been set up properly.
08	xxxx0819	**Equate Symbol**: ISGENQRsn_BadResTable **Meaning**: The RESTABLE specified is not valid. **Action**: Ensure that the resource table does not specify mutually exclusive parameters.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx081A	**Equate Symbol**: ISGENQRsn_BadENQTokenTblAddress **Meaning**: Unable to access the ENQToken table. **Action**: Ensure that the entire ENQToken table is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the ENQToken table length is correct. Ensure that the storage is in the same key as the caller. Note: The ISGENQ request might not have completed.
08	xxxx081B	**Equate Symbol**: ISGENQRsn_BadENQTokenTblALET **Meaning**: Bad ENQToken table ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the ENQToken table is valid. Its access register may not have been set up properly. Note: The ISGENQ request might not have completed.
08	xxxx081C	**Equate Symbol**: ISGENQRsn_BadReturnTableAddress **Meaning**: Unable to access the return table. **Action**: Ensure that the entire return table is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Check that the return table length is correct. Ensure that the storage is in the same key as the caller. Note: The ISGENQ request might not have completed.
08	xxxx081D	**Equate Symbol**: ISGENQRsn_BadReturnTableALET **Meaning**: Bad return table ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the return table is valid. Its access register may not have been set up properly. Note: The ISGENQ request might not have completed.
08	xxxx081E	**Equate Symbol**: ISGENQRsn_NotAuthorizedForQName **Meaning**: For REQUEST=OBTAIN. An unauthorized caller specified an authorized QNAME. **Action**: Unauthorized callers must avoid specifying the authorized QNAMEs listed in the ISGENQ macro prologue.
08	xxxx081F	**Equate Symbol**: ISGENQRsn_NotAuthorizedForExitQname **Meaning**: For REQUEST=OBTAIN. An ISGNQXIT exit specified an authorized QNAME for an unauthorized OBTAIN request. **Action**: Contact your system programmer. The system programmer should check the ISGNQXIT installation exits to ensure they are not coded to specify an authorized QNAME for unauthorized requests.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0821	**Equate Symbol**: ISGENQRsn_NotAuthorizedForOWNINGTTOKEN **Meaning**: An unauthorized caller specified OWNINGTTOKEN. **Action**: Unauthorized callers should avoid specifying OWNINGTTOKEN.
08	xxxx0822	**Equate Symbol**: ISGENQRsn_BadUserDataAddress **Meaning**: Unable to access the USERDATA. **Action**: Ensure that the entire USERDATA is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller.
08	xxxx0823	**Equate Symbol**: ISGENQRsn_BadUserDataAlet **Meaning**: Bad UserData ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the userdata is valid. Its access register may not have been set up properly.
08	xxxx0824	**Equate Symbol**: ISGENQRsn_DeviceNotAllocated **Meaning**: For REQUEST=OBTAIN with RESERVEVOLUME=YES. An unauthorized caller specified a device that is not allocated to the requesting task. **Action**: Unauthorized callers should allocate the UCB to the job step before ISGENQ RESERVEVOLUME(YES) is issued.
08	xxxx0825	**Equate Symbol**: ISGENQRsn_ExitDeviceNotAllocated **Meaning**: For REQUEST=OBTAIN. An ISGNQXIT exit specified a UCB for a device that is not allocated to the requesting, unauthorized task. **Action**: Contact your system programmer. The system programmer should ensure that the installation exits do not modify the UCB to specify one that is not allocated to an unauthorized requests.
08	xxxx0826	**Equate Symbol**: ISGENQRsn_BadControl **Meaning**: Bad CONTROL keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0827	**Equate Symbol**: ISGENQRsn_BadExitUCB@ **Meaning**: The storage pointed to by the UCB address changed by a dynamic exit does not map to a valid UCB. **Action**: Contact your system programmer. The system programmer should ensure that the installation exits do not specify a bad UCB address.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0828	**Equate Symbol**: ISGENQRsn_NotAuthorizedForENQMAX **Meaning**: For REQUEST=OBTAIN, an unauthorized caller specified ENQMAX=NO. **Action**: Unauthorized callers should avoid specifying ENQMAX=NO.
0C	—	**Equate Symbol**: ISGENQRc_EnvError **Meaning**: ISGENQ request has an environment error. **Action**: Refer to action under the individual reason code.
0C	xxxx0C01	**Equate Symbol**: ISGENQRsn_RequestLimitExceeded **Meaning**: For REQUEST=OBTAIN, the limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. **Action**: Retry the request one or more times. If the problem persists, consult your system programmer. For more information on concurrent count limits and how the system can be tuned when necessary, see *z/OS MVS Planning: Global Resource Serialization*.
0C	xxxx0C05	**Equate Symbol**: ISGENQRsn_AbendInExit **Meaning**: One of the GRS dynamic exits abended. **Action**: Retry the request one or more times. Contact your system programmer.
0C	xxxx0C0A	**Equate Symbol**: ISGENQRsn_TaskEnding **Meaning**: The task represented by the specified TToken was ending. The point was reached in task termination after which no ENQs can be obtained. **Action**: Determine why the task identified by the TToken was ending. Correct that error and retry the request.
0C	xxxx0C0B	**Equate Symbol**: ISGENQRsn_FRRHeld **Meaning**: The caller issued ISGENQ when an FRR was established. **Action**: Avoid issuing ISGENQ when using functional recovery routines.
0C	xxxx0C0C	**Equate Symbol**: ISGENQRsn_LockHeld **Meaning**: A lock was held upon entry. No locks can be held when calling ISGENQ. **Action**: Avoid using ISGENQ when locks are held.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
0C	xxxx0C0D	**Equate Symbol**: ISGENQRsn_SrbMode **Meaning**: ISGENQ was issued while in SRB mode. **Action**: Avoid using ISGENQ in SRB mode.
0C	xxxx0C0E	**Equate Symbol**: ISGENQRsn_NotEnabled **Meaning**: ISGENQ was issued while not enabled. **Action**: Avoid using ISGENQ when not enabled.
0C	xxxx0C0F	**Equate Symbol**: ISGENQRsn_MasidTarget **Meaning**: The requester to be released is still the target of an ENQ with the MASID and MTCB options specified. The release does complete and the resource might be damaged. **Action**: The task that issued the ENQ macro instruction with MASID and MTCB should issue the DEQ before this requester does so.
0C	xxxx0C10	**Equate Symbol**: ISGENQRsn_UnsupportedMode. **Meaning**: The current GRS mode does not support this specific request. **Action**: Defer the usage of this particular type of request.
0C	xxxx0C11	**Equate Symbol**: ISGENQRsn_MasidNotSupported. **Meaning**: The resource that was the target of this REQUEST=CHANGE,CONTROL=SHARED request currently or at one time contained MASID users. REQUEST=CHANGE,CONTROL=SHARED is not supported for resources that involve MASID requestors. **Action**: Do not use REQUEST=CHANGE,CONTROL=SHARED on resources that involve MASID requestors.
10	—	**Equate Symbol**: ISGENQRc_CompError **Meaning**: Component Error. **Action**: Contact the IBM Support Center. Reason code that are not defined below contain internal diagnostic information.
10	xxxx1002	**Equate Symbol**: ISGENQRsn_CannotObtainHomeStorage **Meaning**: ISGENQ processing could not obtain storage in the home address space.
10	xxxx1003	**Equate Symbol**: ISGENQRsn_CannotObtainCommonStorage **Meaning**: ISGENQ processing could not obtain storage in the common area.
10	xxxx1004	**Equate Symbol**: ISGENQRsn_CannotObtainPrimaryAlet **Meaning**: ISGENQ processing could not obtain the ALET of the caller's primary address space.

Table 112. Return and Reason Codes for the ISGENQ Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
10	xxxx1006	**Equate Symbol**: ISGENQRsn_SynchResFlushFailed **Meaning**: For REQUEST=OBTAIN, a synchronous reserve failed device state transition flushing.
10	xxxx1007	**Equate Symbol**: ISGENQRsn_ReserveStartFailed **Meaning**: For REQUEST=OBTAIN, reserve start processing failed.
10	xxxx1008	**Equate Symbol**: ISGENQRsn_ReserveCountOverflow **Meaning**: For REQUEST=OBTAIN, reserve processing detected an overflow when updating the reserve count.
10	xxxx1009	**Equate Symbol**: ISGENQRsn_CannotObtainDSQE **Meaning**: ISGENQ processing could not obtain a DSQE to suspend a request during an RNL change.
10	xxxx100A	**Equate Symbol**: ISGENQRsn_ReserveDoneFailed **Meaning**: For REQUEST=OBTAIN, synchronous reserve back-end processing has failed; therefore, the reserve was never completed.
10	xxxx100B	**Equate Symbol**: ISGENQRsn_CannotObtainPrimaryStorage **Meaning**: ENQ/DEQ processing could not obtain storage in the primary address space.

Examples

Use these examples as a guide.

```
* ********************************************************************
*  Request exclusive control of a single resource
* ********************************************************************

         ISGENQ REQUEST=OBTAIN,QNAME=QNAM1,RNAME=RNAM1,RNAMELEN=RLEN1,  X
               SCOPE=SYSTEMS,CONTROL=EXCLUSIVE,ENQTOKEN=ENQT1

* ********************************************************************
*  Release control of a single resource
* ********************************************************************

         ISGENQ REQUEST=RELEASE,ENQTOKEN=ENQT1,COND=YES,               X
               RETCODE=(3),RSNCODE=(2)

* ********************************************************************
*  Conditionally request shared control of 3 resources
* ********************************************************************

         ISGENQ REQUEST=OBTAIN,RESLIST=YES,NUMRES=3,RESTABLE=RSTBL,     X
               ENQTOKENTBL=ETTBL,RETURNTABLE=RTTBL,COND=YES,            X
               RETCODE=(3),RSNCODE=(2),PLISTVER=1

QNAM1    DC    CL8'QNAME1'
RNAM1    DC    CL10'RNAME1'
RLEN1    DC    AL1(L'RNAM1)
RNAM2    DC    CL12'RNAME2'
RNAM3    DC    CL14'RNAME3'
         DS    0D
RSTBL    DS    0CL(3*ISGYENQRES_LEN)
ENTRY1   DC    CL8'QNAME1'          QNAME
         DC    F'0'                 FIRST WORD OF RNAME ADDR
         DC    A(RNAM1)             RNAME ADDR31
```

```
         DC    F'0'                RNAME ALET
         DC    A(0)                UCB@
         DC    AL1(L'RNAM1)        RNAME LENGTH
         DC    AL1(ISGYENQ_kSTEP)
         DC    AL1(ISGYENQ_kCONTROLSHARED)
         DC    XL1'00'             FLAGS
         DC    XL4'00'             RESERVED
ENTRY2   DC    CL8'QNAME2'         QNAME
         DC    F'0'                FIRST WORD OF RNAME ADDR
         DC    A(RNAM2)            RNAME ADDR31
         DC    F'0'                RNAME ALET
         DC    A(0)                UCB@
         DC    AL1(L'RNAM2)        RNAME LENGTH
         DC    AL1(ISGYENQ_kSYSTEM)
         DC    AL1(ISGYENQ_kCONTROLSHARED)
         DC    XL1'00'             FLAGS
         DC    XL4'00'             RESERVED
ENTRY3   DC    CL8'QNAME3'         QNAME
         DC    F'0'                FIRST WORD OF RNAME ADDR
         DC    A(RNAM3)            RNAME ADDR31
         DC    F'0'                RNAME ALET
         DC    A(0)                UCB@
         DC    AL1(L'RNAM3)        RNAME LENGTH
         DC    AL1(ISGYENQ_kSYSTEMS)
         DC    AL1(ISGYENQ_kCONTROLSHARED)
         DC    XL1'00'             FLAGS
         DC    XL4'00'             RESERVED

DYNAREA  DSECT
ENQT1    DS    CL(ISGYENQTOKEN_LEN)
ETTBL    DS    CL(3*ISGYENQTOKEN_LEN)
RTTBL    DS    CL(3*ISGYENQRETURN_LEN)
* ******************************************************************
* Request exclusive control of a single resource with userdata
* ******************************************************************
         ISGENQ REQUEST=OBTAIN,QNAME=QNAM1,RNAME=RNAM1,RNAMELEN=RLEN1, X
               SCOPE=SYSTEMS,CONTROL=EXCLUSIVE,ENQTOKEN=ENQT1,         X
               USERDATA=UDATA1

UDATA1  DC    CL32'MY USERDATA'
```

For more information on global resource serialization, see *z/OS MVS Planning: Global Resource Serialization*.

Chapter 127. ISGLCRT — Create a latch set

Description

Call the Latch_Create service to create a set of latches. Your application should call Latch_Create during application initialization, and specify a number of latches that is sufficient to serialize all the resources that the application requires. Programs that run as part of the application can call the following related services:

ISGLOBT
Requests exclusive or shared ownership of a latch.

ISGLREL
Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPRG
Purges all granted and pending requests for a particular requestor within a specific latch set.

ISGLID
Provides a latch set creator the ability to attach a latch identity array to the latch set to identify the individual latches.

In the following description of Latch_Create, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Create.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you call the Latch_Create service, include the ISGLMASM macro to obtain assembler declaration statements for Latch_Create. ISGLMASM provides the following equate symbols for use when calling Latch_Create:

```
* Latch Create Option Equate Symbols
*
ISGLCRT_PRIVATE               EQU   0
*
* Latch Create Return Codes
*
ISGLCRT_SUCCESS               EQU   0
ISGLCRT_DUPLICATE_NAME        EQU   4
ISGLCRT_NO_STORAGE            EQU   16
*
```

Restrictions

You cannot create a latch set in the master scheduler address space if the master scheduler address space is not also the home address space.

Input register information

Before calling the Latch_Create service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 72-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- number_of_latches
- latch_set_name
- create_option

Latch_Create returns values in the following parameters:

- latch_set_token
- return_code

Syntax	Description
CALL ISGLCRT	,(number_of_latches ,latch_set_name ,create_option ,latch_set_token ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

number_of_latches
Specifies a fullword integer that indicates the number of latches to be created.

,latch_set_name
Specifies a 48-byte area that contains the name of the latch set. The latch set name must be unique within the current address space. The latch set name can be any value up to 48 characters, but the first character must not be binary zeros or an EBCDIC blank. If the latch set name is less than 48 characters, it must be padded on the right with blanks.

IBM recommends that you use a standard naming convention for the latch set name. To avoid using a name that IBM uses, do not begin the latch set name with the character string **SYS.** It is a good idea to select a latch set name that is readable in output from the DISPLAY GRS command and interactive problem control system (IPCS). Avoid '@', '$', and '#' because those characters do not always display consistently.

,create_option
Specifies a fullword integer that must have one of the following values:

- ISGLCRT_PRIVATE (or a value of 0)
- ISGLCRT_PRIVATE + ISGLCRT_LOWSTGUSAGE (or a value of 2)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 (or a value of 64)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 (or a value of 128)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 + ISGLCRT_LOWSTGUSAGE (or a value of 66)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 + ISGLCRT_LOWSTGUSAGE (or a value of 130)

If the creating address space is constrained by private storage, use the ISGLCRT_LOWSTGUSAGE option. ISGLCRT_LOWSTGUSAGE reduces storage usage at the cost of performance. IBM suggests that this option is only used if there is a known or possible storage constraint issue. See "Specifying the Number of Latches in a Latch Set" in *z/OS MVS Programming: Authorized Assembler Services Guide* for a description of the amount of storage that can be consumed by a latch set.

If you want to have the latch obtain services detect some "simple" latch deadlock situations, consider using the ISGLCRT_DEADLOCKDET1 and ISGLCRT_DEADLOCKDET2 options. For performance reasons, latch deadlock detection is not exhaustive. It can detect some simple deadlock situations.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 is specified, it can detect the following deadlock situations:

- The work unit requests exclusive ownership of a latch that the work unit already owns exclusively.
- The work unit requests shared ownership of a latch that the work unit already owns exclusively.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 is specified, it can detect all the deadlock situations listed under ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1, and it can also detect the following situations:

- if the work unit holding a SHARED latch requests exclusive use of the same latch.
- if the work unit holding a SHARED latch requests it SHARED and another unit of work is waiting to obtain the latch EXCLUSIVE.

Because ISGLCRT_DEADLOCKDET2 provides the best deadlock detection, IBM suggests that you use ISGLCRT_DEADLOCKDET1 in cases where it can be used and use ISGLCRT_DEADLOCKDET2 in all cases where there are not many SHARED latch holders.

Note:

1. The unit of work context of the requester is captured at latch obtain time. The system does not know if the application passes responsibility for releasing the latch to another unit of work. To prevent false detection, dead lock detection cannot be used if latches are used in such a way that responsibility for releasing the latch is passed between the obtainer and the releaser.
2. Deadlock detection can be safely used by SRBs, if all the obtained latches are released by the SRB work unit before the unit of work completes. There is a possibility of false deadlock hits otherwise.
3. Deadlock detection is not performed if the latches are obtained conditionally using the ISGLOBT_ASYNC_ECB option in ISGLOBT.

,latch_set_token
Specifies an 8-byte area to contain the latch set token returned by the Latch_Create service. The latch set token uniquely identifies the latch set. Programs must specify this value on calls to the Latch_Obtain, Latch_Release, and Latch_Purge services.

,return_code
A fullword integer to contain the return code from the Latch_Create service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the Latch_Create service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 113. ISGLCRT Return Codes

Return code and Equate symbol	Meaning and Action
00 (0) ISGLCRT_SUCCESS	**Meaning**: The Latch_Create service completed successfully. **Action**: None required.
04 (4) ISGLCRT_DUPLICATE_NAME	**Meaning**: The specified latch_set_name already exists, and is associated with a latch set that was created by a program running in the current primary address space. The latch manager does not create a new latch set. **Action**: To create a new latch set, specify a unique name on the latch_set_name parameter, then call the Latch_Create service again. Otherwise, continue processing with the returned latch set token.
10 (16) ISGLCRT_NO_STORAGE	**Meaning**: Environmental error. Not enough storage was available to contain the requested number of latches. The latch manager does not create a new latch set. **Action**: Specify a smaller value on the number_of_latches parameter.

LATCHX31 - How to call AMODE 31 latch devices

```
         TITLE 'LATCHX31 - How to call AMODE 31 Latch Services'
*** START OF SPECIFICATIONS ***************************************
*
```

```
*01* MODULE-NAME = LATCHX31
*
*02*   DESCRIPTIVE-NAME = SAMPLE PROGRAM WHICH CONTAINS CALLS
*                         TO EACH LATCH SERVICE.
*
*01*PROPRIETARY STATEMENT =
*
*  LICENSED MATERIALS - PROPERTY OF IBM
*  THIS MACRO IS "RESTRICTED MATERIALS OF IBM"
*
*01*STATUS = HBB7760
*
*01* FUNCTION:
*
*    This module provides samples of how to call the following AMODE 31
*    services: ISGLCRT (Latch Create), ISGLID (Latch Id), ISGLOBT
*    (Latch Obtain), ISGLREL (Latch Release), ISGLPRG (Latch Purge),
*    and ISGLPBA (Latch Purge By Address Space).
*
*********************************************************************
*
*02*   RECOVERY-OPERATIONS: None.
*
*********************************************************************
*
*01* NOTES =
*
*  (1) Also shows sample of how to allocate an ISGYLID_ENTRY block
*      to change the Latch ID field of a latch in the latchset.
*
*02*   DEPENDENCIES: None
*
*
*02*   RESTRICTIONS: None
*
*
*02*   REGISTER-CONVENTIONS:
*
*03*        REGISTERS SAVED:    R0-R15
*
*03*        REGISTERS RESTORED: R2-R14
*
*03*        CODE REGISTER:      R12
*
*03*        DATA REGISTER:      R13
*
*02*   PATCH-LABEL: None
*
*01* MODULE-TYPE: Procedure
*
*02*   PROCESSOR:    HLASM
*
*02*   MODULE-SIZE: See External Symbol Dictionary
*
*02*   ATTRIBUTES:
*
*03*      LOCATION:       User specified
*03*      LOAD MODULE:    LATCHX31
*03*      TYPE:           Non-Reentrant
*03*      RMODE:          Any
*03*      SYSGEN:         None
*
*********************************************************************
*
*01* ENTRY-POINT:   LATCHX31
*
*02*   PURPOSE:   See FUNCTION section for this module.
*
*03*     OPERATION:   See FUNCTION section for this module.
*
*03*     ENTRY
*
*04*       MODE:           Enabled
*04*       STATE:          Problem
*04*       KEY:            8
*04*       AMODE:          31
*04*       LOCKS HELD:     None
*04*       ASCMODE:        Primary
*04*       MEMORY MODE:    Non-XMEM
*04*       DISPATCH MODE:  Task
*04*       RECOVERY TYPE:  None
*04*       ADDRESS SPACE:  Caller's
```

```
*
*03*     EXECUTION
*
*04*       MODE:           Enabled
*04*       STATE:          Supervisor
*04*       KEY:            0
*04*       AMODE:          31
*04*       LOCKS OBTAINED: None
*04*       ASCMODE:        Primary
*04*       MEMORY MODE:    Non-XMEM
*04*       ADDRESS SPACE:  Caller's
*
*02*   LINKAGE: Branched to.
*
*03*     CALLERS:
*
*         Any
*
*02*   INPUT:
*
*03*     ENTRY-REGISTERS:
*
*           R0 - R12,R15 - Irrelevant
*           AR0-AR15     - Irrelevant
*
*02*   OUTPUT:
*
*02*   EXIT-NORMAL: RETURN TO CALLER
*
*03*     CONDITIONS: successful completion
*
*03*     EXIT-REGISTERS:
*
*           R0  -R14  - Unchanged
*           R15       - Return code (always 0)
*           AR0-AR15  - Irrelevant
*
*03*     RETURN-CODES:  None
*
*     Return code    Reason code     Interpretation
*     -----------    -----------     --------------
*         '0'x           N/A         Success
*
*02*   EXIT-ERROR:   None
*
*********************************************************************
*
*01* TERMINATION-CONSIDERATIONS:  None
*
*********************************************************************
*
*01* EXTERNAL-REFERENCES :
*
*
*02*   ROUTINES:  Latch Services.
*
*02*   DATA-AREAS:   None
*
*02*   CONTROL-BLOCKS:
*
*      Name    Use  Mapping  Description
*      ------- ---- -------- ----------
*      ISGYLID CW   ISGLMASM Latch Identity Entry
*
*      Legend:  C=Create, R=Read, W=Write, D=Delete
*
*01* TABLES:  TRTABLE
*
*01* MACROS-EXECUTABLE:
*
*         None
*
*01* SERIALIZATION:
*
*         None
*
*01* MESSAGES:
*
*         None
*
*01* POST-CODES:
*
```

```
*         None
*
*01* ABEND-CODES:
*
*         None
*
*01* WAIT-STATE-CODES:
*
*         None
*
*01* CHANGE-ACTIVITY:
*
*         None
*
**** END OF SPECIFICATIONS *********************************************
LATCHX31 CSECT
LATCHX31 AMODE 31
LATCHX31 RMODE ANY
         BAKR  R14,R0              Save gprs 2-14 and PSW
         SAC   0                   Ensure primary mode
         BRAS  R12,PSTART          Establish addressability
PSTART   EQU   *
         USING PSTART,12
         MODESET MODE=SUP          Get into supervisor state
         STORAGE OBTAIN,LENGTH=DYNALEN Get savearea and dynamic area
         LR    R13,R1              Place savearea address into reg13
         USING DYNASTORE,R13
         MVC   4(4,R13),=C'F1SA'   Set the Save area ID (31 bit)
*
*********************************************************************
* Create latch set
*********************************************************************
*
         CALL ISGLCRT,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),      X
               MF=(E,CREATE_DPL)
*
*********************************************************************
*
* Initialize Storage for initial LID Entry Block.
* Note that in this example the DYNASTOR section is not freed and
* non-pertinant data is placed in the DYNASTOR section for simplicity.
*
* The default subpool associated with the STORAGE OBTAIN macro has a
* lifetime of the address space, so the DYNASTOR * section will exist
* for the life of the address space as well.
*
* However, IBM recommends using a separate storage request
* for the LIDPointerArray and ISGYLID_Entry blocks so that only
* necessary data will continue to exist beyond the life of the calling
* module.
*
* Also, it is not necessary to initialize all latch IDs in the set
* to point to a default ISGYLID_ENTRY block. NULL values in the
* Latch ID Pointer Array are acceptable.
*
*********************************************************************
*
         XC    INIT_STOR,INIT_STOR
         XC    ENTIRELIDARRAY,ENTIRELIDARRAY
         LA    R3,INIT_STOR        Base ISGYLID_ENTRY block            X
                                   at address of allocated storage
         LLGTR R3,R3               Clear high half of 64-bit address
         USING ISGYLID_ENTRY,R3
         LA    R4,INITLIDSTR
         LLGTR R4,R4               Clear high half of 64-bit address
         STG   R4,LIDPRINTABLESTRING@
         LHI   R4,L'INITLIDSTR     Put length of INITLIDSTR into       X
                                   entry block
         STH   R4,LIDPRINTABLESTRINGLENGTH
         L     R4,ONEMINUTE        Set hold threshold value to         X
                                   one minute
         ST    R4,LIDHOLDTHRESHOLD
         L     R4,THIRTYSECONDS    Set contention threshold value to   X
                                   thirty seconds
         ST    R4,LIDCONTTHRESHOLD
*
*********************************************************************
*
* Set Latch ID array entries all to address of Initial Lid Entry ---
* indicating all entries are currently unused.
*
* Latch ID Pointer Array must be in the primary address space.
```

```
*
*********************************************************************
*
         LHI   R6,0
         L     R4,NUM_LATCH
LOOP1    EQU   *
         STG   R3,LIDPTRARRAY(R6)
         A     R6,LIDPTRARRAYLEN
         BCT   R4,LOOP1
         DROP  R3
*
*********************************************************************
*
* Attach Latch IDs to Latch Set
*
*********************************************************************
*
         CALL ISGLID,(LS_TOKEN,LIDPTRARRAY,LIDVERSION,                X
               RETCD),MF=(E,LID_DPL)
*
*********************************************************************
*
* Change Latch ID Entry for latch #3
*
* Note:
* Once the Latch Identity Pointer Array has been attached to the
* latch set, it cannot be deleted. However it can be replaced by
* calling the service again and specifying a new array. To change
* the Latch Identity for a particular latch, allocate a new latch
* identity block, fill it out, and update the corresponding array
* entry. Then the program is free to delete the previous latch
* identity block.
*
*********************************************************************
*
         MVC   NEW_STOR,INIT_STOR  Copy LID Entry to new storage
         USING ISGYLID_ENTRY,R3    Use R3 as base for LID Entry Block
         LA    R3,NEW_STOR
         LLGTR R3,R3               Clearing high half of 64-bit address
         LA    R4,NEWLIDSTR
         LLGTR R4,R4               Clearing high half of 64-bit address
         STG   R4,LIDPRINTABLESTRING@ Store address and length of new  X
                                      string in ISGYLID_ENTRY block
         LHI   R5,L'NEWLIDSTR
         STH   R5,LIDPRINTABLESTRINGLENGTH
         L     R4,LATCH_NUM        Point to new ISGLID_ENTRY block
         SLL   R4,3                Multiply by 8 (size of LIDPTRARRAY)
         STG   R3,LIDPTRARRAY(R4)
         DROP  R3
*
*********************************************************************
* Obtain latch #3
*********************************************************************
*
         MVC   REQ_IDH,PSAAOLD-PSA Use ASCB address as high half
         MVC   REQ_IDL,PSATOLD-PSA Use TCB address as low half
         CALL ISGLOBT,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE,    X
               ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD),                X
               MF=(E,OBTAIN_DPL)
*
*********************************************************************
* Release latch #3
*********************************************************************
*
         CALL ISGLREL,(LS_TOKEN,LATCH_TOKEN1,                         X
               UNCOND,WORK_AREA,RETCD),MF=(E,RELEASE_DPL)
*
*********************************************************************
*
* Purge requestor from latch set.
*
* Normally reserved for recovery situations.
*
*********************************************************************
*
         XC    REQ_IDL,REQ_IDL     Clear the low half of requestor ID
         CALL ISGLPRG,(LS_TOKEN,REQ_ID,RETCD),MF=(E,PURGE_DPL)
*
*********************************************************************
*
* Purge all granted and pending requests for a group of requestors for
* a group of latch sets in the current address space.
```

```
*
* Normally reserved for recovery situations.
*
*********************************************************************
*
         CALL ISGLPBA,(=AD(0),REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETCD),  X
               MF=(E,PURGEBA_DPL)
*
*********************************************************************
* Exit
*
* Restore caller's regs and return. Also restores caller's PSW key and
* State without a MODESET MODE=PROB.
*
*********************************************************************
*
         LHI   R15,0               Set return code to zero
         PR ,
*
*********************************************************************
* Equates
*********************************************************************
*
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
*
*
*********************************************************************
* Constants
*********************************************************************
*
NUM_LATCH    DC  F'16'             Number of latches to create - input
*                                    to create
LS_NAME      DC  CL48'EXAMPLE.ONE_LATCH_SET' latch set name - input to
*                                    create
LS_MASK      DC  48XL1'FF'         Latch set name mask to match all
*                                    names - input to purge group
REQ_MASK     DC  4XL1'FF'
             DC  4XL1'00'          Requestor ID mask to match all
*                                    requestors with the same first
*                                    half (ascb@) and any second half
INITLIDSTR   DC  C'Latch not used' Initial LID string and length
*
LIDVERSION   DC  AL1(ISGYLID_VERSION1) Set latch version to default
*
LIDPTRARRAYLEN DC F'8'
*
THIRTYSECONDS DC F'30'
*
ONEMINUTE    DC  F'60'
*
NEWLIDSTR    DC  C'Serializing Abstract Resource' New string and length
*
PRIVATE      DC  A(ISGLCRT_PRIVATE) Create option - input to create -
*                                    (defined in IDF)
LATCH_NUM    DC  F'3'              Number of latch to be obtained -
*                                    input to obtain
SUSPEND      DC  A(ISGLOBT_SYNC)   Obtain option - input to obtain -
*                                    (defined in IDF)
EXCLUSIVE    DC  A(ISGLOBT_EXCLUSIVE) access option - input to obtain -
*                                    (defined in IDF)
UNCOND       DC  A(ISGLREL_UNCOND) Release option - input to release -
*                                    (defined in IDF)
*********************************************************************
* Dynamic area for save area, parmlists, and variables
*********************************************************************
*
DYNASTORE    DSECT
```

```
*
SAVEAREA     DS  18F              72-byte register save area
*
ENTIRELIDARRAY DS 0XL128
*
LIDPTRARRAY  DS  16AD             LIDPtrArray
*
CREATE_DPL   DS  0F
   CALL ,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),MF=L
*
OBTAIN_DPL   DS  0F
   CALL ,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE,                X
               ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD),MF=L
*
LID_DPL      DS  0F
   CALL ,(LS_TOKEN,LIDPTRARRAY,LIDVERSION,                            X
               RETCD),MF=L
*
RELEASE_DPL  DS  0F
   CALL ,(LS_TOKEN,LATCH_TOKEN1,UNCOND,WORK_AREA,RETCD),MF=L
*
PURGE_DPL    DS  0F
   CALL ,(LS_TOKEN,REQ_ID,RETCD),MF=L
*
PURGEBA_DPL  DS  0F
   CALL ,(0,REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETCD),MF=L
*
ECB          DS  F                ECB (used only when the obtain
*                                   option is ISGLOBT_ASYNC_ECB)
ECB_ADDR     DS  A                Address of ECB - input to obtain -
*                                   (required for the interface, but
*                                   only used when obtain option is
*                                   ISGLOBT_ASYNC_ECB)
LS_TOKEN     DS  2F               Latch set token - output from create
*                                   and input to obtain, latch identity,
*                                   release, purge, and purge group
LATCH_TOKEN1 DS  2F               Latch token - output from obtain
*                                   and input to release
REQ_ID       DS  0D               Double word requestor ID - input to
*                                   obtain, purge, and purge group
REQ_IDH      DS  F                First half of requestor ID (ascb@)
*
REQ_IDL      DS  F                Second half of requestor ID (tcb@)
*
WORK_AREA    DS  32D              256-byte work area
*
RETCD        DS  F                Return code - output from services
*
INIT_STOR    DS  XL(ISGYLID_ENTRY_LEN) Isgylid_entry block storage
*
NEW_STOR     DS  XL(ISGYLID_ENTRY_LEN)
*
DYNALEN      EQU *-DYNASTORE
*
             IHAPSA ,             Needed for ascb@ and tcb@ req ID
             ISGLMASM ,           Needed for latch constants
             END LATCHX31
```

Chapter 128. ISGLCR64 — Create a latch set in 64-bit mode

Description

Call the 64-bit Latch_Create service to create a set of latches. Your application should call Latch_Create during application initialization, and specify a number of latches that is sufficient to serialize all the resources that the application requires. Programs that run as part of the application can call the following related services:

ISGLOB64
Requests exclusive or shared ownership of a latch.

ISGLRE64
Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPR64
Purges all granted and pending requests for a particular requestor within a specific latch set.

ISGLID64
Provides a latch set creator the ability to attach a latch identity array to the latch set to identify the individual latches.

In the following description of 64-bit Latch_Create, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Create.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you call the 64-bit Latch_Create service, include the ISGLMASM macro to obtain assembler declaration statements for 64-bit Latch_Create. ISGLMASM provides the following equate symbols for use when calling Latch_Create:

```
* Latch Create Option Equate Symbols
*
ISGLCRT_PRIVATE               EQU   0
*
* Latch Create Return Codes
*
ISGLCRT_SUCCESS               EQU   0
ISGLCRT_DUPLICATE_NAME        EQU   4
```

```
ISGLCRT_NO_STORAGE            EQU   16
*
```

Restrictions

You cannot create a latch set in the master scheduler address space if the master scheduler address space is not also the home address space.

Input register information

Before calling the 64-bit Latch_Create service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 144-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- number_of_latches
- latch_set_name
- create_option

The 64-bit Latch_Create returns values in the following parameters:

- latch_set_token
- return_code

Syntax	Description
CALL ISGLCR64	,(number_of_latches ,latch_set_name ,create_option ,latch_set_token ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

number_of_latches
Specifies a fullword integer that indicates the number of latches to be created.

,latch_set_name
Specifies a 48-byte area that contains the name of the latch set. The latch set name must be unique within the current address space. The latch set name can be any value up to 48 characters, but the first character must not be binary zeros or an EBCDIC blank. If the latch set name is less than 48 characters, it must be padded on the right with blanks.

IBM recommends that you use a standard naming convention for the latch set name. To avoid using a name that IBM uses, do not begin the latch set name with the character string **SYS.** It is a good idea to select a latch set name that is readable in output from the DISPLAY GRS command and interactive problem control system (IPCS). Avoid '@', '$', and '#' because those characters do not always display consistently.

,create_option
Specifies a fullword integer that must have one of the following values:

- ISGLCRT_PRIVATE (or a value of 0)
- ISGLCRT_PRIVATE + ISGLCRT_LOWSTGUSAGE (or a value of 2)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 (or a value of 64)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 (or a value of 128)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 + ISGLCRT_LOWSTGUSAGE (or a value of 66)
- ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 + ISGLCRT_LOWSTGUSAGE (or a value of 130)

If the creating address space is constrained by private storage, use the ISGLCRT_LOWSTGUSAGE option. ISGLCRT_LOWSTGUSAGE reduces storage usage at the cost of performance. IBM suggests that this option is only used if there is a known or possible storage constraint issue. See "Specifying the Number of Latches in a Latch Set" in *z/OS MVS Programming: Authorized Assembler Services Guide* for a description of the amount of storage that can be consumed by a latch set.

If you want to have the latch obtain services detect some "simple" latch deadlock situations, consider using the ISGLCRT_DEADLOCKDET1 and ISGLCRT_DEADLOCKDET2 options. For performance reasons, latch deadlock detection is not exhaustive. It can detect some simple deadlock situations.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1 is specified, it can detect the following deadlock situations:

- The work unit requests exclusive ownership of a latch that the work unit already owns exclusively.
- The work unit requests shared ownership of a latch that the work unit already owns exclusively.

When ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET2 is specified, it can detect all the deadlock situations listed under ISGLCRT_PRIVATE + ISGLCRT_DEADLOCKDET1, and it can also detect if the work unit holding a SHARED latch requests exclusive use of the same latch. It also catches multiple Share requests by same unit of work when there is exclusive Waiter in between -> Shared (UW1) - Exclusive (UW2) - Shared (UW1).

Because ISGLCRT_DEADLOCKDET2 provides the best deadlock detection, IBM suggests that you use ISGLCRT_DEADLOCKDET1 in cases where it can be used and use ISGLCRT_DEADLOCKDET2 in all cases where there are not many SHARED latch holders.

Note:

1. The unit of work context of the requester is captured at latch obtain time. The system does not know if the application passes responsibility for releasing the latch to another unit of work. To prevent false detection, dead lock detection cannot be used if latches are used in such a way that responsibility for releasing the latch is passed between the obtainer and the releaser.
2. Deadlock detection can be safely used by SRBs, if all the obtained latches are released by the SRB work unit before the unit of work completes. There is a possibility of false deadlock hits otherwise.
3. Deadlock detection is not performed if the latches are obtained conditionally using the ISGLOBT_ASYNC_ECB option in ISGLOBT.

,latch_set_token
Specifies an 8-byte area to contain the latch set token returned by the Latch_Create service. The latch set token uniquely identifies the latch set. Programs must specify this value on calls to the Latch_Obtain, Latch_Release, and Latch_Purge services.

,return_code
A fullword integer to contain the return code from the Latch_Create service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the Latch_Create service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 114. ISGLCR64 Return Codes

Return code and Equate symbol	Meaning and Action
00 (0) ISGLCRT_SUCCESS	**Meaning**: The Latch_Create service completed successfully. **Action**: None required.
04 (4) ISGLCRT_DUPLICATE_NAME	**Meaning**: The specified latch_set_name already exists, and is associated with a latch set that was created by a program running in the current primary address space. The latch manager does not create a new latch set. **Action**: To create a new latch set, specify a unique name on the latch_set_name parameter, then call the Latch_Create service again. Otherwise, continue processing with the returned latch set token.
10 (16) ISGLCRT_NO_STORAGE	**Meaning**: Environmental error. Not enough storage was available to contain the requested number of latches. The latch manager does not create a new latch set. **Action**: Specify a smaller value on the number_of_latches parameter.

LATCHX64 - How to call AMODE 64 latch services

```
         TITLE 'LATCHX64 - How to call AMODE 64 Latch Services'
*** START OF SPECIFICATIONS *****************************************
*
*01* MODULE-NAME = LATCHX64
*
*02*   DESCRIPTIVE-NAME = SAMPLE PROGRAM WHICH CONTAINS CALLS
*                        TO EACH LATCH SERVICE.
*
*01*PROPRIETARY STATEMENT =
*
*  LICENSED MATERIALS - PROPERTY OF IBM
*  THIS MACRO IS "RESTRICTED MATERIALS OF IBM"
*
*01*STATUS = HBB7760
*
*01* FUNCTION:
*
*    This module provides samples of how to call the following AMODE 64
*    services: ISGLCR64 (Latch Create), ISGLID64 (Latch Id), ISGLOB64
*    (Latch Obtain), ISGLRE64 (Latch Release), ISGLPR64 (Latch Purge),
*    and ISGLPB64 (Latch Purge By Address Space).
*
*********************************************************************
*
*02*   RECOVERY-OPERATIONS: None.
*
*********************************************************************
*
*01* NOTES =
*
*  (1) Also shows sample of how to allocate an ISGYLID_ENTRY block
*      to change the Latch ID field of a latch in the latchset.
*
*02*   DEPENDENCIES: None
*
*
*02*   RESTRICTIONS: None
*
*
*02*   REGISTER-CONVENTIONS:
*
*03*        REGISTERS SAVED:    R0-R15
*
*03*        REGISTERS RESTORED: R2-R14
*
*03*        CODE REGISTER:      R12
*
*03*        DATA REGISTER:      R13
*
*02*   PATCH-LABEL: None
*
*01* MODULE-TYPE: Procedure
*
*02*   PROCESSOR:    HLASM
*
*02*   MODULE-SIZE: See External Symbol Dictionary
*
*02*   ATTRIBUTES:
*
*03*      LOCATION:       User specified
*03*      LOAD MODULE:    LATCHX64
*03*      TYPE:           Non-Reentrant
*03*      RMODE:          Any
*03*      SYSGEN:         None
*
*********************************************************************
*
*01* ENTRY-POINT:   LATCHX64
*
*02*   PURPOSE:   See FUNCTION section for this module.
*
*03*     OPERATION:   See FUNCTION section for this module.
*
*03*     ENTRY
*
*04*       MODE:           Enabled
*04*       STATE:          Problem
*04*       KEY:            8
```

```
*04*       AMODE:          64
*04*       LOCKS HELD:     None
*04*       ASCMODE:        Primary
*04*       MEMORY MODE:    Non-XMEM
*04*       DISPATCH MODE:  Task
*04*       RECOVERY TYPE:  None
*04*       ADDRESS SPACE:  Caller's
*
*03*     EXECUTION
*
*04*       MODE:           Enabled
*04*       STATE:          Supervisor
*04*       KEY:            0
*04*       AMODE:          64
*04*       LOCKS OBTAINED: None
*04*       ASCMODE:        Primary
*04*       MEMORY MODE:    Non-XMEM
*04*       ADDRESS SPACE:  Caller's
*
*02*   LINKAGE: Branched to.
*
*03*     CALLERS:
*
*         Any
*
*02*   INPUT:
*
*03*     ENTRY-REGISTERS:
*
*           R0 - R12,R15 - Irrelevant
*           AR0-AR15     - Irrelevant
*
*02*   OUTPUT:
*
*02*   EXIT-NORMAL: RETURN TO CALLER
*
*03*     CONDITIONS: successful completion
*
*03*     EXIT-REGISTERS:
*
*           R0  -R14  - Unchanged
*           R15       - Return code (always 0)
*           AR0-AR15  - Irrelevant
*
*03*     RETURN-CODES:  None
*
*     Return code    Reason code     Interpretation
*     -----------    -----------     --------------
*         '0'x           N/A         Success
*
*02*   EXIT-ERROR:   None
*
*********************************************************************
*
*01* TERMINATION-CONSIDERATIONS:  None
*
*********************************************************************
*
*01* EXTERNAL-REFERENCES :
*
*
*02*   ROUTINES:  Latch Services.
*
*02*   DATA-AREAS:   None
*
*02*   CONTROL-BLOCKS:
*
*      Name    Use  Mapping  Description
*      ------- ---- -------- ----------
*      ISGYLID CW   ISGLMASM Latch Identity Entry
*
*      Legend:  C=Create, R=Read, W=Write, D=Delete
*
*01* TABLES:  TRTABLE
*
*01* MACROS-EXECUTABLE:
*
*         None
*
*01* SERIALIZATION:
*
*         None
```

```
*
*01* MESSAGES:
*
*         None
*
*01* POST-CODES:
*
*         None
*
*01* ABEND-CODES:
*
*         None
*
*01* WAIT-STATE-CODES:
*
*         None
*
*01* CHANGE-ACTIVITY:
*
*         None
*
**** END OF SPECIFICATIONS *********************************************
LATCHX64 CSECT
LATCHX64 AMODE 64
LATCHX64 RMODE ANY
         SYSSTATE AMODE64=YES      Indicate AMODE 64
         BAKR  R14,R0              Save gprs 2-14 and PSW
         SAC   0                   Ensure primary mode
         BRAS  R12,PSTART          Establish addressability
PSTART   EQU   *
         USING PSTART,12
         MODESET MODE=SUP          Get into supervisor state
         STORAGE OBTAIN,LENGTH=DYNALEN Get savearea and dynamic area
         LGR   R13,R1              Place savearea address into reg13
         USING DYNASTORE,R13
         MVC   4(4,R13),=C'F4SA'   Set the Save area ID (64 bit)
*
*********************************************************************
* Create latch set
*********************************************************************
*
         CALL ISGLCR64,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),     X
               MF=(E,CREATE_DPL)
*
*********************************************************************
*
* Initialize Storage for initial LID Entry Block.
* Note that in this example the DYNASTOR section is not freed and
* non-pertinent data is placed in the DYNASTOR section for simplicity.
*
* The default subpool associated with the STORAGE OBTAIN macro has a
* lifetime of the address space, so the DYNASTOR * section will exist
* for the life of the address space as well.
*
* However, IBM recommends using a separate storage request
* for the LIDPointerArray and ISGYLID_Entry blocks so that only
* necessary data will continue to exist beyond the life of the calling
* module.
*
* Also, it is not necessary to initialize all latch IDs in the set
* to point to a default ISGYLID_ENTRY block. NULL values in the
* Latch ID Pointer Array are acceptable.
*
*********************************************************************
*
         XC    INIT_STOR,INIT_STOR
         XC    ENTIRELIDARRAY,ENTIRELIDARRAY
         LA    R3,INIT_STOR        Base ISGYLID_ENTRY block            X
                                   at address of allocated storage
         USING ISGYLID_ENTRY,R3
         LA    R4,INITLIDSTR
         STG   R4,LIDPRINTABLESTRING@
         LHI   R4,L'INITLIDSTR     Put length of INITLIDSTR into       X
                                   entry block
         STH   R4,LIDPRINTABLESTRINGLENGTH
         L     R4,ONEMINUTE        Set hold threshold value to         X
                                   one minute
         ST    R4,LIDHOLDTHRESHOLD
         L     R4,THIRTYSECONDS    Set contention threshold value to   X
                                   thirty seconds
         ST    R4,LIDCONTTHRESHOLD
*
```

```
*********************************************************************
*
* Set Latch ID array entries all to address of Initial Lid Entry ---
* indicating all entries are currently unused.
*
* Latch ID Pointer Array must be in the primary address space.
*
*********************************************************************
*
         LGHI  R6,0
         L     R4,NUM_LATCH
LOOP1    EQU   *
         STG   R3,LIDPTRARRAY(R6)
         A     R6,LIDPTRARRAYLEN
         BCT   R4,LOOP1
         DROP  R3
*
*********************************************************************
*
* Attach Latch IDs to Latch Set
*
*********************************************************************
*
         CALL ISGLID64,(LS_TOKEN,LIDPTRARRAY,LIDVERSION,               X
               RETCD),MF=(E,LID_DPL)
*
*********************************************************************
*
* Change Latch ID Entry for latch #3
*
* Note:
* Once the Latch Identity Pointer Array has been attached to the
* latch set, it cannot be deleted. However it can be replaced by
* calling the service again and specifying a new array. To change
* the Latch Identity for a particular latch, allocate a new latch
* identity block, fill it out, and update the corresponding array
* entry. Then the program is free to delete the previous latch
* identity block.
*
*********************************************************************
*
         MVC   NEW_STOR,INIT_STOR  Copy LID Entry to new storage
         USING ISGYLID_ENTRY,R3    Use R3 as base for LID Entry Block
         LA    R3,NEW_STOR
         LA    R4,NEWLIDSTR
         STG   R4,LIDPRINTABLESTRING@ Store address and length of new  X
                                      string in ISGYLID_ENTRY block
         LHI   R5,L'NEWLIDSTR
         STH   R5,LIDPRINTABLESTRINGLENGTH
         L     R4,LATCH_NUM        Point to new ISGLID_ENTRY block
         SLL   R4,3                Multiply by 8 (size of LIDPTRARRAY)
         STG   R3,LIDPTRARRAY(R4)
         DROP  R3
*
*********************************************************************
* Obtain latch #3
*********************************************************************
*
         MVC   REQ_IDH,PSAAOLD-PSA Use ASCB address as high half
         MVC   REQ_IDL,PSATOLD-PSA Use TCB address as low half
         CALL ISGLOB64,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE,   X
               ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD),                 X
               MF=(E,OBTAIN_DPL)
*
*********************************************************************
* Release latch #3
*********************************************************************
*
         CALL ISGLRE64,(LS_TOKEN,LATCH_TOKEN1,                         X
               UNCOND,WORK_AREA,RETCD),MF=(E,RELEASE_DPL)
*
*********************************************************************
*
* Purge requestor from latch set.
*
* Normally reserved for recovery situations.
*
*********************************************************************
*
         XC    REQ_IDL,REQ_IDL     Clear the low half of requestor ID
         CALL ISGLPR64,(LS_TOKEN,REQ_ID,RETCD),MF=(E,PURGE_DPL)
*
```

```
*********************************************************************
*
* Purge all granted and pending requests for a group of requestors for
* a group of latch sets in the current address space.
*
* Normally reserved for recovery situations.
*
*********************************************************************
*
         CALL ISGLPB64,(=AD(0),REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETCD), X
               MF=(E,PURGEBA_DPL)
*
*********************************************************************
* Exit
*
* Restore caller's regs and return. Also restores caller's PSW key and
* State without a MODESET MODE=PROB.
*
*********************************************************************
*
         LGHI  R15,0               Set return code to zero
         PR ,
*
*********************************************************************
* Equates
*********************************************************************
*
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
*
*
*********************************************************************
* Constants
*********************************************************************
*
NUM_LATCH    DC  F'16'             Number of latches to create - input
*                                    to create
LS_NAME      DC  CL48'EXAMPLE.ONE_LATCH_SET' latch set name - input to
*                                    create
LS_MASK      DC  48XL1'FF'         Latch set name mask to match all
*                                    names - input to purge group
REQ_MASK     DC  4XL1'FF'
             DC  4XL1'00'          Requestor ID mask to match all
*                                    requestors with the same first
*                                    half (ascb@) and any second half
INITLIDSTR   DC  C'Latch not used' Initial LID string and length
*
LIDVERSION   DC  AL1(ISGYLID_VERSION1) Set latch version to default
*
LIDPTRARRAYLEN DC F'8'
*
THIRTYSECONDS DC F'30'
*
ONEMINUTE    DC  F'60'
*
NEWLIDSTR    DC  C'Serializing Abstract Resource' New string and length
*
PRIVATE      DC  A(ISGLCRT_PRIVATE) Create option - input to create -
*                                    (defined in IDF)
LATCH_NUM    DC  F'3'              Number of latch to be obtained -
*                                    input to obtain
SUSPEND      DC  A(ISGLOBT_SYNC)   Obtain option - input to obtain -
*                                    (defined in IDF)
EXCLUSIVE    DC  A(ISGLOBT_EXCLUSIVE) access option - input to obtain -
*                                    (defined in IDF)
UNCOND       DC  A(ISGLREL_UNCOND) Release option - input to release -
*                                    (defined in IDF)
*********************************************************************
```

```
* Dynamic area for save area, parmlists, and variables
*********************************************************************
*
DYNASTORE    DSECT
*
SAVEAREA     DS  18D              144-byte register save area
*
ENTIRELIDARRAY DS 0XL128
*
LIDPTRARRAY  DS  16AD             LIDPtrArray
*
CREATE_DPL   DS  0F
   CALL ,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),MF=L
*
OBTAIN_DPL   DS  0F
   CALL ,(LS_TOKEN,LATCH_NUM,REQ_ID,SUSPEND,EXCLUSIVE,                X
               ECB_ADDR,LATCH_TOKEN1,WORK_AREA,RETCD),MF=L
*
LID_DPL      DS  0F
   CALL ,(LS_TOKEN,LIDPTRARRAY,LIDVERSION,                            X
               RETCD),MF=L
*
RELEASE_DPL  DS  0F
   CALL ,(LS_TOKEN,LATCH_TOKEN1,UNCOND,WORK_AREA,RETCD),MF=L
*
PURGE_DPL    DS  0F
   CALL ,(LS_TOKEN,REQ_ID,RETCD),MF=L
*
PURGEBA_DPL  DS  0F
   CALL ,(0,REQ_ID,REQ_MASK,LS_NAME,LS_MASK,RETCD),MF=L
*
ECB          DS  F                ECB (used only when the obtain
*                                   option is ISGLOBT_ASYNC_ECB)
ECB_ADDR     DS  A                Address of ECB - input to obtain -
*                                   (required for the interface, but
*                                   only used when obtain option is
*                                   ISGLOBT_ASYNC_ECB)
LS_TOKEN     DS  2F               Latch set token - output from create
*                                   and input to obtain, latch identity,
*                                   release, purge, and purge group
LATCH_TOKEN1 DS  2F               Latch token - output from obtain
*                                   and input to release
REQ_ID       DS  0D               Double word requestor ID - input to
*                                   obtain, purge, and purge group
REQ_IDH      DS  F                First half of requestor ID (ascb@)
*
REQ_IDL      DS  F                Second half of requestor ID (tcb@)
*
WORK_AREA    DS  64D              512-byte work area
*
RETCD        DS  F                Return code - output from services
*
INIT_STOR    DS  XL(ISGYLID_ENTRY_LEN) Isgylid_entry block storage
*
NEW_STOR     DS  XL(ISGYLID_ENTRY_LEN)
*
DYNALEN      EQU *-DYNASTORE
*
             IHAPSA ,             Needed for ascb@ and tcb@ req ID
             ISGLMASM ,           Needed for latch constants
             END LATCHX64
```

Chapter 129. ISGLID — Identify a latch set

Description

The ISGLID callable service provides a latch set creator the ability to attach a latch identity array (in ISGLMASM or ISGLMC) to the latch set for the purposes of identifying the individual latches in the latch set. The LIDArray must be in the primary space. The following callable services are related to the ISGLID service:

- ISGLCRT
- ISGLCRT64
- ISGLID
- ISGLID64

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Primary= the space of the latch set creator
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

After the latch identity pointer array has been attached to the latch set, the attached LIDArray cannot be deleted. However, the LIDArray can be replaced by calling the service again and specifying a new array. To change the latch identity for a particular latch, allocate a new latch identity block, fill it out, and update the corresponding array entry. Then, you can delete the previous latch identity block.

Restrictions

None.

Input register information

Before calling the ISGLID service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 72-byte save area located in the primary address space.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14-15
Used as work registers by the system.

Performance implications

None

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- lsetToken
- LIDArray
- LIDEntryVersion

ISGLID returns values in the following parameter:

- retcode

Syntax	Description
CALL ISGLID	,(lsetToken ,LIDPtrArray ,LIDEntryVersion ,retcode)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

lsetToken
Specifies an 8-character field that contains the latch set token returned from ISGLCRT.

LIDPtrArray
Specifies the latch identity pointer array.

LIDEntryVersion
Specifies a 1-byte area that contains the version of the LID entries.

RetCode
Specifies a 4-byte or 32-bit area that contains return code from the ISGLID service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses for these codes.

Return codes

When the ISGLID service returns control to your program, the RetCode parameter contains a hexadecimal return code. The following table identifies the hexadecimal reason codes and meaning associated with each reason code:

Table 115. ISGLID Return Codes

Return code and Equate symbol	Meaning and action
00000000 ISGLID_SUCCESS	**Meaning**: The ISGLID service completed successfully. **Action**: None.
xxxx0401 ISGLID_REPLACED	**Meaning**: Latch identity pointer array is replaced. A previous latch identity pointer array existed for this latch set. It has been replaced. **Action**: None.

Example

See “LATCHX31 - How to call AMODE 31 latch devices” on page 1240.

Chapter 130. ISGLID64 — Identify a latch set in 64-bit mode

Description

The ISGLID64 callable service provides a latch set creator the ability to attach a latch identity array (in ISGLMASM or ISGLMC) to the latch set for the purposes of identifying the individual latches in the latch set. The LIDArray must be in the primary space. The following callable services are related to the ISGLID64 service:

- ISGLCRT
- ISGLCRT64
- ISGLID
- ISGLID64

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Primary= the space of the latch set creator
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

After the latch identity pointer array has been attached to the latch set, the attached LIDArray cannot be deleted. However, the LIDArray can be replaced by calling the service again and specifying a new array. To change the latch identity for a particular latch, allocate a new latch identity block, fill it out, and update the corresponding array entry. Then, you can delete the previous latch identity block.

Restrictions

None.

Input register information

Before calling the ISGLID64 service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 144-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- lsetToken
- LIDArray
- LIDEntryVersion

ISGLID64 returns values in the following parameter:

- retcode

Syntax	Description
CALL ISGLID64	,(lsetToken ,LIDPtrArray ,LIDEntryVersion ,retcode)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

lsetToken
Specifies an 8-character field that contains the latch set token returned from ISGLCRT.

LIDPtrArray
Specifies the latch identity pointer array.

LIDEntryVersion
Specifies a 1-byte area that contains the version of the LID entries.

RetCode
Specifies a 4-byte or 32-bit area that contains return code from the ISGLID64 service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses for these codes.

Return codes

When the ISGLID64 service returns control to your program, the RetCode parameter contains a hexadecimal return code. The following table identifies the hexadecimal reason codes and meaning associated with each reason code:

Table 116. ISGLID64 Return Codes

Return code and Equate symbol	Meaning and action
00000000	**Meaning**: The ISGLID service completed successfully. **Action**: None.
xxxx0401	**Meaning**: Latch identity pointer array is replaced. A previous latch identity pointer array existed for this latch set. It has been replaced. **Action**: None.

Example

See “LATCHX64 - How to call AMODE 64 latch services” on page 1251.

Chapter 131. ISGLOBT — Obtain a latch

Description

Call the Latch_Obtain service to request exclusive or shared ownership of a latch. When a requestor owns a particular latch, the requestor can use the resource associated with that latch. The following callable services are related to Latch_Obtain:

ISGLCRT
Creates a latch set that an application can use to serialize resources.

ISGLREL
Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPRG
Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Obtain:

- The term *requestor* describes a task or SRB routine that calls the Latch_Obtain service to request ownership of a latch.
- Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, "ISGLOBT_COND (value of 1)" indicates the equate symbol ISGLOBT_COND and its associated value, 1.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming Requirements

If you specify an obtain_option of ISGLOBT_ASYNC_ECB (as described in “Parameters” on page 1267), initialize the ECB pointed to by the value on the ECB_address parameter to zero before calling Latch_Obtain.

Before you use the Latch_Obtain service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Obtain. ISGLMASM provides the following equate symbols for use when calling Latch_Obtain:

```
*
* Latch Obtain Option Equate Symbols
*
ISGLOBT_SYNC                  EQU   0
ISGLOBT_COND                  EQU   1
```

```
ISGLOBT_ASYNC_ECB             EQU   2
*
*   Latch Obtain Access Equate Symbols
*
ISGLOBT_EXCLUSIVE             EQU   0
ISGLOBT_SHARED                EQU   1
*
*   Latch Obtain Equate Symbols
*
ISGLOBT_SUCCESS               EQU   0
ISGLOBT_CONTENTION            EQU   4
*
```

Restrictions

1. The caller of Latch_Obtain must have a PSW key that allows access to the latch set storage.
2. The ECB specified on the ECB_address parameter must reside in storage with a storage key that matches the latch set storage key.
3. You must call Latch_Obtain from the same primary address space from which the Latch_Create service was called.

Input register information

Before calling the Latch_Obtain service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 72-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

See the information about obtaining latches in *z/OS MVS Programming: Authorized Assembler Services Guide* for performance implications related to the Latch_Obtain service.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- latch_set_token
- latch_number
- requestor_ID
- obtain_option
- access_option
- ECB_address

Latch_Obtain returns values in the following parameters:

- latch_token
- return_code

Latch_Obtain uses the following parameter for temporary storage:

- work_area

Syntax	Description
CALL ISGLOBT	,(latch_set_token ,latch_number ,requestor_ID ,obtain_option ,access_option ,ECB_address ,latch_token ,work_area ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token that the Latch_Create service returned earlier when it created the latch set.

,latch_number
Specifies a fullword integer that contains the number of the latch to be obtained. The latch_number must be in the range from 0 to the total number of latches in the associated latch set minus one.

,requestor_ID
Specifies an 8-byte area that contains a value that identifies the caller of the Latch_Obtain service. The requestor_ID can be any value except all binary zeros.

Recovery routines can purge all granted and pending requests for a particular requestor (identified by a requestor_id) within a specific latch set. When specifying the requestor_ID on Latch_Obtain, consider which latches would be purged if the Latch_Purge service were to be called with the specified requestor_ID. For more information about the Latch_Purge service, see Chapter 135, “ISGLPRG — Purge a requestor from a latch set,” on page 1285.

,obtain_option
A fullword integer that specifies how the system is to handle the Latch_Obtain request if the latch manager cannot immediately grant ownership of the latch to the requestor:

ISGLOBT_SYNC (value of 0)
The system processes the request synchronously. The system suspends the requestor. When the latch manager eventually grants ownership of the latch to the requestor, the system returns control to the requestor.

ISGLOBT_COND (value of 1)
The system processes the request conditionally. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). The latch manager does not queue the request to obtain the latch.

ISGLOBT_ASYNC_ECB (value of 2)
The system processes the request asynchronously. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). When the latch manager eventually grants ownership of the latch to the requestor, the system posts the ECB pointed to by the value specified on the ECB_address parameter.

When you specify this option, the ECB_address parameter must contain the address of an initialized ECB that is addressable from the home address space (HASN).

,access_option
A fullword or character string that specifies the access required:

- ISGLOBT_EXCLUSIVE (value of 0) - Exclusive (write) access
- ISGLOBT_SHARED (value of 1) - Shared (read) access

,ECB_address
Specifies a fullword that contains the address of an ECB. If you specify an obtain_option of ISGLOBT_SYNC (value of 0) or ISGLOBT_COND (value of 1) on the call to Latch_Obtain, the ECB_address field must be valid (though its contents are ignored). IBM recommends that an address of 0 be used when no ECB is to be processed.

If you specify an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) and the system returns a return code of ISGLOBT_CONTENTION (value of 4) to the caller, the system posts the ECB pointed to by the value specified on the ECB_address parameter when the latch manager grants ownership of the latch to the requestor.

,latch_token
Specifies an 8-byte area to contain the latch token returned by the Latch_Obtain service. You must provide this value as a parameter on a call to the Latch_Release service to release the latch.

,work_area
Specifies a 256-byte work area that provides temporary storage for the Latch_Obtain service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Obtain.

,return_code
Specifies a fullword integer that is to contain the return code from the Latch_Obtain service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses for these codes.

Return codes

When the Latch_Obtain service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 117. ISGLOBT Return Codes	
Return code and Equate Symbol	**Meaning and Action**
00 (0) ISGLOBT_SUCCESS	**Meaning**: The Latch_Obtain service completed successfully. **Action**: None.
04 (4) ISGLOBT_CONTENTION	**Meaning**: A requestor called Latch_Obtain with an obtain_option of ISGLOBT_COND (value of 1) or ISGLOBT_ASYNC_ECB (value of 2). The latch is not immediately available. **Action**: If the requestor specified an obtain_option of ISGLOBT_COND (value of 1), no response is required. If the requestor specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2), and the latch is still required, wait on the ECB to be posted when the latch manager grants ownership of the latch to the requestor.

Example

See “LATCHX31 - How to call AMODE 31 latch devices” on page 1240 for an example of how to call Latch_Obtain in assembler language.

Chapter 132. ISGLOB64 — Obtain a latch in 64-bit mode

Description

Call the 64-bit Latch_Obtain service to request exclusive or shared ownership of a latch. When a requestor owns a particular latch, the requestor can use the resource associated with that latch. The following callable services are related to Latch_Obtain:

ISGLCR64
Creates a latch set that an application can use to serialize resources.

ISGLRE64
Releases ownership of an owned latch or a pending request to obtain a latch.

ISGLPR64
Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of 64-bit Latch_Obtain:

- The term *requestor* describes a task or SRB routine that calls the Latch_Obtain service to request ownership of a latch.
- Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, "ISGLOBT_COND (value of 1)" indicates the equate symbol ISGLOBT_COND and its associated value, 1.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

If you specify an obtain_option of ISGLOBT_ASYNC_ECB (as described in "Parameters" on page 1273), initialize the ECB pointed to by the value on the ECB_address parameter to zero before calling Latch_Obtain.

Before you use the Latch_Obtain service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Obtain. ISGLMASM provides the following equate symbols for use when calling Latch_Obtain:

```
*
* Latch Obtain Option Equate Symbols
```

```
*
ISGLOBT_SYNC                  EQU   0
ISGLOBT_COND                  EQU   1
ISGLOBT_ASYNC_ECB             EQU   2

*
*   Latch Obtain Access Equate Symbols
*
ISGLOBT_EXCLUSIVE             EQU   0
ISGLOBT_SHARED                EQU   1
*
*   Latch Obtain Equate Symbols
*
ISGLOBT_SUCCESS               EQU   0
ISGLOBT_CONTENTION            EQU   4
*
```

Restrictions

1. The caller of the 64-bit Latch_Obtain must have a PSW key that allows access to the latch set storage.
2. The ECB specified on the ECB_address parameter must reside in storage with a storage key that matches the latch set storage key.
3. You must call the 64-bit Latch_Obtain from the same primary address space from which the 64-bitLatch_Create service was called.

Input register information

Before calling the 64-bit Latch_Obtain service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 144-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

See the information about obtaining latches in *z/OS MVS Programming: Authorized Assembler Services Guide* for performance implications related to the 64-bit Latch_Obtain service.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- latch_set_token
- latch_number
- requestor_ID
- obtain_option
- access_option
- ECB_address

The 64-bit Latch_Obtain returns values in the following parameters:

- latch_token
- return_code

The 64-bit Latch_Obtain uses the following parameter for temporary storage:

- work_area

Syntax	Description
CALL ISGLOB64	,(latch_set_token ,latch_number ,requestor_ID ,obtain_option ,access_option ,ECB_address ,latch_token ,work_area ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token that the 64-bit Latch_Create service returned earlier when it created the latch set.

,latch_number
Specifies a fullword integer that contains the number of the latch to be obtained. The latch_number must be in the range from 0 to the total number of latches in the associated latch set minus one.

,requestor_ID
Specifies an 8-byte area that contains a value that identifies the caller of the 64-bit Latch_Obtain service. The requestor_ID can be any value except all binary zeros.

Recovery routines can purge all granted and pending requests for a particular requestor (identified by a requestor_id) within a specific latch set. When specifying the requestor_ID on the 64-bit Latch_Obtain, consider which latches would be purged if the 64-bit Latch_Purge service were to be called with the specified requestor_ID. For more information about the 64-bit Latch_Purge service, see Chapter 136, “ISGLPR64 — Purge a requestor from a latch set in 64-bit mode,” on page 1289.

,obtain_option
A fullword integer that specifies how the system is to handle the 64-bit Latch_Obtain request if the latch manager cannot immediately grant ownership of the latch to the requestor:

ISGLOB64_SYNC (value of 0)
The system processes the request synchronously. The system suspends the requestor. When the latch manager eventually grants ownership of the latch to the requestor, the system returns control to the requestor.

ISGLOB64_COND (value of 1)
The system processes the request conditionally. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). The latch manager does not queue the request to obtain the latch.

ISGLOB64_ASYNC_ECB (value of 2)
The system processes the request asynchronously. The system returns control to the requestor with a return code of ISGLOBT_CONTENTION (value of 4). When the latch manager eventually grants ownership of the latch to the requestor, the system posts the ECB pointed to by the value specified on the ECB_address parameter.

When you specify this option, the ECB_address parameter must contain the address of an initialized ECB that is addressable from the home address space (HASN).

,access_option
A fullword or character string that specifies the access required:

- ISGLOBT_EXCLUSIVE (value of 0) - Exclusive (write) access
- ISGLOBT_SHARED (value of 1) - Shared (read) access

,ECB_address
Specifies a fullword that contains the address of an ECB. If you specify an obtain_option of ISGLOB64 _SYNC (value of 0) or ISGLOBT_COND (value of 1) on the call to Latch_Obtain, the ECB_address field must be valid (though its contents are ignored). IBM recommends that an address of 0 be used when no ECB is to be processed.

If you specify an obtain_option of ISGLOB64 _ASYNC_ECB (value of 2) and the system returns a return code of ISGLOBT_CONTENTION (value of 4) to the caller, the system posts the ECB pointed to by the value specified on the ECB_address parameter when the latch manager grants ownership of the latch to the requestor.

,latch_token
Specifies an 8-byte area to contain the latch token returned by the 64-bit Latch_Obtain service. You must provide this value as a parameter on a call to the 64-bit Latch_Release service to release the latch.

,work_area
Specifies a 512-byte work area that provides temporary storage for the 64-bit Latch_Obtain service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of 64-bit Latch_Obtain.

,return_code
Specifies a fullword integer that is to contain the return code from the 64-bit Latch_Obtain service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses for these codes.

Return codes

When the 64-bit Latch_Obtain service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 118. ISGLOBT64 Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLOBT_SUCCESS	**Meaning**: The Latch_Obtain service completed successfully. **Action**: None.
04 (4) ISGLOBT_CONTENTION	**Meaning**: A requestor called Latch_Obtain with an obtain_option of ISGLOBT_COND (value of 1) or ISGLOBT_ASYNC_ECB (value of 2). The latch is not immediately available. **Action**: If the requestor specified an obtain_option of ISGLOBT_COND (value of 1), no response is required. If the requestor specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2), and the latch is still required, wait on the ECB to be posted when the latch manager grants ownership of the latch to the requestor.

Example

See “LATCHX64 - How to call AMODE 64 latch services” on page 1251 for an example of how to call Latch_Obtain in assembler language.

Chapter 133. ISGLPBA — Purge a group of requestors from a group of latch sets

Description

Call the Latch_Purge_by_Address_Space service to purge all granted and pending requests for a group of requestors for a group of latch sets in the same address space. To effectively use this service, your latch_set_names and your requestor_IDs should be defined such that they have a common portion and a unique portion. Groups of latch sets can then be formed by masking off the unique portion of the latch_set_name, and groups of latch requests in a latch set can then be formed by masking off the unique portion of the requestor_ID. Masking off the unique portion of the requestor_ID allows a single purge request to handle multiple latch sets and multiple requests in a latch set. Recovery routines should call Latch_Purge_by_Address_Space when one or more errors prevent requestors from releasing latches.

The following callable services are related to Latch_Purge_by_Address_Space:

ISGLCRT
Creates a latch set that an application can use to serialize resources.

ISGLOBT
Requests exclusive or shared control of a latch.

ISGLREL
Releases control of an owned latch or a pending request to obtain a latch.

ISGLPRG
Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Purge_by_Address_Space, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Purge_by_Address_Space.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you use the Latch_Purge_by_Address_Space service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Purge_by_Address_Space. ISGLMASM provides the following equate symbols for use when calling Latch_Purge_by_Address_Space

```
*
*   Latch Purge by Address Space Return Codes
*
ISGLPRG_SUCCESS               EQU   0
ISGLPRG_DAMAGE_DETECTED       EQU   4
ISGLPRG_INCORRECT_MASK        EQU   C
*
```

Restrictions

1. The caller of Latch_Purge_by_Address_Space must have a PSW key that allows access to the latch set storage.
2. You must call Latch_Purge_by_Address_Space from the same primary address space from which the Latch_Create service was called.

Input register information

Before calling the Latch_Purge_by_Address_Space service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 72-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- latch_set_token
- requestor_ID
- requestor_ID_mask
- latch_set_name
- latch_set_name_mask

Latch_Purge_by_Address_Space returns a value in the return_code parameter.

Syntax	Description
CALL ISGLPBA	,(latch_set_token ,requestor_ID ,requestor_ID_mask ,latch_set_name ,latch_set_name_mask ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service or a value of zero. If the value is not zero, the latch_set_token identifies the latch set from which latch requests are to be purged. If the latch_set_token is set to zero, a group of latch sets, determined by the latch_set_name and latch_set_name_mask, will have their latch requests purged.

,requestor_id
Specifies an 8-byte area that contains a portion of the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. This operand will be compared to the result of logically ANDing each requestor_ID in the latch set with the requestor_ID_mask. Make sure that any corresponding bits that are zero in the requestor_ID_mask are also zero in this field, otherwise no ID matches will occur. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

,requestor_id_mask
Specifies an 8-byte area that contains the requestor_ID_mask that will be logically ANDed to each requestor_ID in the latch set and then compared to the requestor_ID operand. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

,latch_set_name
Specifies a 48-byte area that contains the portion of the latch_set_name that will be compared to the result of logically ANDing the latch_set_name_mask with each latch set name in the primary address space. Make sure that any corresponding bits that are zero in the latch_set_name_mask are also zero in this field, otherwise no name matches will occur. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

,latch_set_name_mask
Specifies a 48-byte area that contains the mask that will be logically ANDed to each of the latch set names in the primary address apace and then compared to the latch_set_name operand. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

,return_code
A fullword integer that contains the return code from the Latch_Purge_By_Address_Space service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the Latch_Purge_by_Address_Space service returns control to your program, the return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 119. ISGLPBA Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	**Meaning**: The Latch_Purge_by_Address_Space service completed successfully. **Action**: None.
04 (4) ISGLPRG_DAMAGE_DETECTED	**Meaning**: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with *correct* data) for the specified requestor. **Action**: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.

Chapter 134. ISGLPB64 — Purge a group of requestors from a group of latch sets in 64-bit mode

Description

Call the 64-bit Latch_Purge_by_Address_Space service to purge all granted and pending requests for a group of requestors for a group of latch sets in the same address space. To effectively use this service, your latch_set_names and your requestor_IDs should be defined such that they have a common portion and a unique portion. Groups of latch sets can then be formed by masking off the unique portion of the latch_set_name, and groups of latch requests in a latch set can then be formed by masking off the unique portion of the requestor_ID. Masking off the unique portion of the requestor_ID allows a single purge request to handle multiple latch sets and multiple requests in a latch set. Recovery routines should call 64-bit Latch_Purge_by_Address_Space when one or more errors prevent requestors from releasing latches.

The following callable services are related to 64-bit Latch_Purge_by_Address_Space:

ISGLCR64
Creates a 64-bit latch set that an application can use to serialize resources.

ISGLOB64
Requests exclusive or shared control of a 64-bit latch.

ISGLRE64
Releases control of an owned 64-bit latch or a pending request to obtain a 64-bit latch.

ISGLPR64
Purges all granted and pending requests for a particular requestor within a specific 64-bit latch set.

In the following description of the 64-bit Latch_Purge_by_Address_Space, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to the 64–bit Latch_Purge_by_Address_Space.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you use the 64–bit Latch_Purge_by_Address_Space service, you need to include the ISGLMASM macro to obtain assembler declaration statements for the 64–bit Latch_Purge_by_Address_Space.

ISGLMASM provides the following equate symbols for use when calling the 64–bit Latch_Purge_by_Address_Space

```
*
*   Latch Purge by Address Space Return Codes
*
ISGLPRG_SUCCESS               EQU   0
ISGLPRG_DAMAGE_DETECTED       EQU   4
ISGLPRG_INCORRECT_MASK        EQU   C
*
```

Restrictions

1. The caller of the 64–bit Latch_Purge_by_Address_Space must have a PSW key that allows access to the latch set storage.
2. You must call the 64–bit Latch_Purge_by_Address_Space from the same primary address space from which the 64–bit Latch_Create service was called.

Input register information

Before calling the 64–bit Latch_Purge_by_Address_Space service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 144-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- latch_set_token
- requestor_ID
- requestor_ID_mask
- latch_set_name
- latch_set_name_mask

The 64–bit Latch_Purge_by_Address_Space returns a value in the return_code parameter.

Syntax	Description
CALL ISGLPB64	,(latch_set_token ,requestor_ID ,requestor_ID_mask ,latch_set_name ,latch_set_name_mask ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service or a value of zero. If the value is not zero, the latch_set_token identifies the latch set from which latch requests are to be purged. If the latch_set_token is set to zero, a group of latch sets, determined by the latch_set_name and latch_set_name_mask, will have their latch requests purged.

,requestor_id
Specifies an 8-byte area that contains a portion of the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. This operand will be compared to the result of logically ANDing each requestor_ID in the latch set with the requestor_ID_mask. Make sure that any corresponding bits that are zero in the requestor_ID_mask are also zero in this field, otherwise no ID matches will occur. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

,requestor_id_mask
Specifies an 8-byte area that contains the requestor_ID_mask that will be logically ANDed to each requestor_ID in the latch set and then compared to the requestor_ID operand. Each requestor_ID that has a name match will have its Latch_Obtain requests released.

,latch_set_name
Specifies a 48-byte area that contains the portion of the latch_set_name that will be compared to the result of logically ANDing the latch_set_name_mask with each latch set name in the primary address space. Make sure that any corresponding bits that are zero in the latch_set_name_mask are also zero in this field, otherwise no name matches will occur. Each latch set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

,latch_set_name_mask
Specifies a 48-byte area that contains the mask that will be logically ANDed to each of the latch set names in the primary address apace and then compared to the latch_set_name operand. Each latch

set that has a name match will have its Latch_Obtain requests released. If the latch_set_token operand is non-zero this operand is ignored.

,return_code
A fullword integer that contains the return code from the Latch_Purge_By_Address_Space service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the 64–bit Latch_Purge_by_Address_Space service returns control to your program, the return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 120. ISGLPB64 Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	**Meaning**: The Latch_Purge_by_Address_Space service completed successfully. **Action**: None.
04 (4) ISGLPRG_DAMAGE_DETECTED	**Meaning**: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with *correct* data) for the specified requestor. **Action**: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.

Chapter 135. ISGLPRG — Purge a requestor from a latch set

Description

Call the Latch_Purge service to purge all granted and pending requests for a particular requestor within a specific latch set. Recovery routines should call Latch_Purge when one or more errors prevent requestors from releasing latches. The following callable services are related to Latch_Purge:

ISGLCRT
Creates a latch set that an application can use to serialize resources.

ISGLOBT
Requests exclusive or shared control of a latch.

ISGLREL
Releases control of an owned latch or a pending request to obtain a latch.

In the following description of Latch_Purge, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Purge.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you use the Latch_Purge service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Purge. ISGLMASM provides the following equate symbols for use when calling Latch_Purge:

```
*
*   Latch Purge Return Codes
*
ISGLPRG_SUCCESS               EQU   0
ISGLPRG_DAMAGE_DETECTED       EQU   4
*
```

Restrictions

1. The caller of Latch_Purge must have a PSW key that allows access to the latch set storage.

2. You must call Latch_Purge from the same primary address space from which the Latch_Create service was called.

Input register information

Before calling the Latch_Purge service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 72-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- latch_set_token
- requestor_ID

Latch_Purge returns a value in the return_code parameter.

Syntax	Description
CALL ISGLPRG	,(latch_set_token ,requestor_ID ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service. The latch set token identifies the latch set from which latch requests are to be purged.

,requestor_ID
Specifies an 8-byte area that contains the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. The Latch_Purge service is to release all Latch_Obtain requests that specify this requestor_ID.

,return_code
A fullword integer that contains the return code from the Latch_Purge service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the Latch_Purge service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 121. ISGLPRG Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	**Meaning**: The Latch_Purge service completed successfully. **Action**: None.
04 (4) ISGLPRG_DAMAGE_DETECTED	**Meaning**: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with *correct* data) for the specified requestor. **Action**: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.

Example

See “LATCHX31 - How to call AMODE 31 latch devices” on page 1240 for an example of how to call Latch_Purge in assembler language.

Chapter 136. ISGLPR64 — Purge a requestor from a latch set in 64-bit mode

Description

Call the 64-bit Latch_Purge service to purge all granted and pending requests for a particular requestor within a specific 64-bit latch set. Recovery routines should call 64-bit Latch_Purge when one or more errors prevent requestors from releasing latches. The following callable services are related to the 64-bit Latch_Purge:

ISGLCR64
Creates a 64-bit latch set that an application can use to serialize resources.

ISGLOB64
Requests exclusive or shared control of a 64-bit latch.

ISGLRE64
Releases control of an owned 64-bit latch or a pending request to obtain a 64-bit latch.

In the following description of 64-bit Latch_Purge, equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to 64-bit Latch_Purge.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you use the 64-bit Latch_Purge service, you need to include the ISGLMASM macro to obtain assembler declaration statements for Latch_Purge. ISGLMASM provides the following equate symbols for use when calling Latch_Purge:

```
*
*   Latch Purge Return Codes
*
ISGLPRG_SUCCESS               EQU   0
ISGLPRG_DAMAGE_DETECTED       EQU   4
*
```

Restrictions

1. The caller of 64-bit Latch_Purge must have a PSW key that allows access to the latch set storage.

2. You must call the 64-bit Latch_Purge from the same primary address space from which the 64-bit Latch_Create service was called.

Input register information

Before calling the 64-bit Latch_Purge service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 144-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Assign values to the following parameters:

- latch_set_token
- requestor_ID

The 64-bit Latch_Purge returns a value in the return_code parameter.

Syntax	Description
CALL ISGLPR64	,(latch_set_token ,requestor_ID ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch_set_token previously returned by the Latch_Create service. The latch set token identifies the latch set from which latch requests are to be purged.

,requestor_ID
Specifies an 8-byte area that contains the requestor_ID originally specified on one or more previous calls to the Latch_Obtain service. The Latch_Purge service is to release all Latch_Obtain requests that specify this requestor_ID.

,return_code
A fullword integer that contains the return code from the Latch_Purge service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the 64-bit Latch_Purge service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 122. ISGLPRG Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLPRG_SUCCESS	**Meaning**: The Latch_Purge service completed successfully. **Action**: None.
04 (4) ISGLPRG_DAMAGE_DETECTED	**Meaning**: Program error. While purging all requests for a particular requestor from a latch set, the latch manager found incorrect data in one or more latches. The latch manager tries to purge the latches that contain incorrect data, but the damage might prevent the latch manager from purging those latches. The latch manager purges the remaining latches (those with *correct* data) for the specified requestor. **Action**: Take a dump and check for a storage overlay. If your application can continue without the resources serialized by the damaged latches, no action is required.

Example

See “LATCHX64 - How to call AMODE 64 latch services” on page 1251 for an example of how to call 64-bit Latch_Purge in assembler language.

Chapter 137. ISGLREL — Release a latch

Description

Call the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch. Requestors should call Latch_Release when the use of a resource associated with a latch is no longer required. The following callable services are related to Latch_Release:

ISGLCRT
Creates a latch set that an application can use to serialize resources.

ISGLOBT
Requests exclusive or shared control of a latch.

ISGLPRG
Purges all granted and pending requests for a particular requestor within a specific latch set.

In the following description of Latch_Release:

- The term *requestor* describes a program that calls the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch.
- Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to Latch_Obtain. For example, "ISGLREL_COND (value of 1)" indicates the equate symbol ISGLREL_COND and its associated value, 1.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you use the Latch_Release service, include the ISGLMASM macro to obtain assembler declaration statements for Latch_Release. ISGLMASM provides the following equate symbols for use when calling Latch_Release:

```
*
*   Latch Release Option Equate Symbols
*
ISGLREL_UNCOND                EQU   0
ISGLREL_COND                  EQU   1
*
*   Latch Release Return Codes
*
ISGLREL_SUCCESS               EQU   0
ISGLREL_NOT_OWNED_ECB_REQUEST EQU   4
```

```
ISGLREL_STILL_SUSPENDED       EQU   8
ISGLREL_INCORRECT_LATCH_TOKEN EQU  12
*
```

Restrictions

1. The caller of Latch_Release must have a PSW key that allows access to the latch set storage.
2. You must call Latch_Release from the same primary address space from which the Latch_Create service was called.

Input register information

Before calling the Latch_Release service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 72-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

See the information about releasing latches in *z/OS MVS Programming: Authorized Assembler Services Guide* for performance implications related to the Latch_Release service.

Syntax

Write the CALL as shown on the syntax diagram, coding all parameters in the specified order.

Assign values to the following parameters:

- latch_set_token
- latch_token
- release_option

Latch_Release returns a value in the following parameter:

- return_code

Latch_Release uses the following parameter for temporary storage:

- work_area

Syntax	Description
CALL ISGLREL	,(latch_set_token ,latch_token ,release_option ,work_area ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch set token returned to the caller of the Latch_Create service. The latch set token identifies the latch set that contains the latch to be released.

,latch_token
Specifies an 8-byte area that contains the latch token returned to the caller of the Latch_Obtain service. The latch token identifies the request to be released.

,release_option
Specifies a fullword integer that tells the latch manager what to do when the requestor either no longer owns the latch to be released or still has a pending request to obtain the latch to be released:

ISGLREL_UNCOND (value of 0)
Abend the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) and is suspended while waiting to obtain the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0009.
- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) and is suspended while waiting to obtain the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0007.
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system abends the caller of Latch_Release with abend X'9C6', reason code xxxx000A.

ISGLREL_COND (value of 1)
Return control to the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) and the latch has been obtained, but the ECB has not been posted, the latch manager releases the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_NOT_OWNED_ECB_REQUEST (value of 4).
- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) but is suspended while waiting to obtain the latch, the latch manager does not release the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_STILL_SUSPENDED (value of 8).
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system returns control to the caller of Latch_Release with a return code of ISGLREL_INCORRECT_LATCH_TOKEN (value of 12).

,work_area
Specifies a 256-byte work area that provides temporary storage for the Latch_Release service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Release.

,return_code
Specifies a fullword integer that is to contain the return code from the Latch_Release service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the Latch_Release service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 123. ISGLREL Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLREL_SUCCESS	**Meaning**: The Latch_Release service completed successfully. The caller released ownership of the specified latch request. **Action**: None.
04 (4) ISGLREL_NOT_OWNED_ECB_REQUEST	**Meaning**: The requestor that originally called the Latch_Obtain service is still expecting the system to post an ECB (to indicate that the requestor has obtained the latch). The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not post the ECB at the address specified on the original call to Latch_Obtain. The latch manager releases the latch. **Action**: Validate the integrity of the resource associated with the latch (the requestor might have used the resource without waiting on the ECB). If the resource is undamaged, no action is necessary (a requestor routine may have been in the process of cancelling the request to obtain the latch).
08 (8) ISGLREL_STILL_SUSPENDED	**Meaning**: Program error. The request specified a correct latch token, but the program that originally requested the latch is still suspended and waiting to obtain the latch. The latch requestor originally specified an obtain_option of ISGLOBT_SYNC on the call to the Latch_Obtain service. The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not release the latch. The latch requestor remains suspended. **Action**: • Wait for the latch requestor to obtain the latch and receive control back from the system; then call the Latch_Release service again, or • End the program that originally requested the latch.

Table 123. ISGLREL Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
0C (12) ISGLREL_INCORRECT_LATCH_TOKEN	**Meaning**: The latch manager could not find a granted or pending request associated with the value on the latch token parameter. The latch manager does not release a latch. This return code does not indicate an error if a routine calls Latch_Release to ensure that a latch is released. For example, if an error occurs when a requestor calls the Latch_Obtain service, the requestor's recovery routine might call Latch_Release to ensure that the requested latch is released. If the error prevented the requestor from obtaining the latch, the recovery routine receives this return code. **Action**: If the return code is not expected, validate that the latch token is the same latch token returned to the caller of Latch_Obtain.

Example

See “LATCHX31 - How to call AMODE 31 latch devices” on page 1240 for an example of how to call Latch_Release in assembler language.

Chapter 138. ISGLRE64 — Release a latch in 64-bit mode

Description

Call the 64-bit Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch. Requestors should call 64-bit Latch_Release when the use of a resource associated with a latch is no longer required. The following callable services are related to 64-bit Latch_Release:

ISGLCR64
Creates a 64-bit latch set that an application can use to serialize resources.

ISGLOB64
Requests exclusive or shared control of a 64-bit latch.

ISGLPR64
Purges all granted and pending requests for a particular requestor within a specific 64-bit latch set.

In the following description of the 64-bit Latch_Release:

- The term *requestor* describes a program that calls the Latch_Release service to release ownership of an owned latch or a pending request to obtain a latch.
- Equate symbols defined in the ISGLMASM macro are followed by their numeric equivalents; you may specify either when coding calls to the 64-bit Latch_Obtain. For example, “ISGLREL_COND (value of 1)” indicates the equate symbol ISGLREL_COND and its associated value, 1.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW allowing key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be accessible from the primary address space.

Programming requirements

Before you use the 64-bit Latch_Release service, include the ISGLMASM macro to obtain assembler declaration statements for Latch_Release. ISGLMASM provides the following equate symbols for use when calling the 64-bit Latch_Release:

```
*
*   Latch Release Option Equate Symbols
*
ISGLREL_UNCOND                EQU   0
ISGLREL_COND                  EQU   1
*
*   Latch Release Return Codes
```

```
*
ISGLREL_SUCCESS               EQU   0
ISGLREL_NOT_OWNED_ECB_REQUEST EQU   4
ISGLREL_STILL_SUSPENDED       EQU   8
ISGLREL_INCORRECT_LATCH_TOKEN EQU  12
*
```

Restrictions

1. The caller of the 64-bit Latch_Release must have a PSW key that allows access to the latch set storage.
2. You must call the 64-bit Latch_Release from the same primary address space from which the 64-bitLatch_Create service was called.

Input register information

Before calling the 64-bit Latch_Release service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 144-byte save area located in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

See the information about releasing latches in *z/OS MVS Programming: Authorized Assembler Services Guide* for performance implications related to the 64-bit Latch_Release service.

Syntax

Write the CALL as shown on the syntax diagram, coding all parameters in the specified order.

Assign values to the following parameters:

- latch_set_token

- latch_token
- release_option

The 64-bit Latch_Release returns a value in the following parameter:

- return_code

The 64-bit Latch_Release uses the following parameter for temporary storage:

- work_area

Syntax	Description
CALL ISGLRE64	,(latch_set_token ,latch_token ,release_option ,work_area ,return_code)

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

latch_set_token
Specifies an 8-byte area that contains the latch set token returned to the caller of the 64-bit Latch_Create service. The latch set token identifies the latch set that contains the latch to be released.

,latch_token
Specifies an 8-byte area that contains the latch token returned to the caller of the 64-bit Latch_Obtain service. The latch token identifies the request to be released.

,release_option
Specifies a fullword integer that tells the latch manager what to do when the requestor either no longer owns the latch to be released or still has a pending request to obtain the latch to be released:

ISGLREL_UNCOND (value of 0)
Abend the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) when obtaining the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0009.
- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) when obtaining the latch, the latch manager does not release the latch. The system abends the caller of Latch_Release with abend X'9C6', reason code xxxx0007.
- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system abends the caller of Latch_Release with abend X'9C6', reason code xxxx000A.

ISGLREL_COND (value of 1)
Return control to the requestor:

- If a requestor originally specified an obtain_option of ISGLOBT_ASYNC_ECB (value of 2) when obtaining the latch, the latch manager releases the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_NOT_OWNED_ECB_REQUEST (value of 4).
- If a requestor originally specified an obtain_option of ISGLOBT_SYNC (value of 0) when obtaining the latch, the latch manager does not release the request for ownership of the latch. The system returns control to the caller of Latch_Release with a return code of ISGLREL_STILL_SUSPENDED (value of 8).

- If the latch manager does not find a previous Latch_Obtain request for the specified latch, the system returns control to the caller of Latch_Release with a return code of ISGLREL_INCORRECT_LATCH_TOKEN (value of 12).

,work_area
Specifies a 512-byte work area that provides temporary storage for the 64-bit Latch_Release service. The work area should begin on a doubleword boundary to optimize performance. The work area must be in the same storage key as the caller of Latch_Release.

,return_code
Specifies a fullword integer that is to contain the return code from the 64-bit Latch_Release service.

ABEND codes

The caller might encounter abend code X'9C6' for certain errors. See *z/OS MVS System Codes* for explanations and responses.

Return codes

When the 64-bit Latch_Release service returns control to your program, return_code contains a hexadecimal return code. The following table identifies return codes in hexadecimal and decimal (in parentheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 124. ISGLRE64 Return Codes

Return code and Equate Symbol	Meaning and Action
00 (0) ISGLREL_SUCCESS	**Meaning**: The Latch_Release service completed successfully. The caller released ownership of the specified latch request. **Action**: None.
04 (4) ISGLREL_NOT_OWNED_ECB_REQUEST	**Meaning**: The requestor that originally called the Latch_Obtain service is still expecting the system to post an ECB (to indicate that the requestor has obtained the latch). The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not post the ECB at the address specified on the original call to Latch_Obtain. The latch manager releases the latch. **Action**: Validate the integrity of the resource associated with the latch (the requestor might have used the resource without waiting on the ECB). If the resource is undamaged, no action is necessary (a requestor routine may have been in the process of cancelling the request to obtain the latch).
08 (8) ISGLREL_STILL_SUSPENDED	**Meaning**: Program error. The request specified a correct latch token, but the program that originally requested the latch is still suspended and waiting to obtain the latch. The latch requestor originally specified an obtain_option of ISGLOBT_SYNC on the call to the Latch_Obtain service. The call to the Latch_Release service specified a release_option of ISGLREL_COND (value of 1). The latch manager does not release the latch. The latch requestor remains suspended. **Action**: • Wait for the latch requestor to obtain the latch and receive control back from the system; then call the Latch_Release service again, or • End the program that originally requested the latch.

Table 124. ISGLRE64 Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
0C (12) ISGLREL_INCORRECT_LATCH_TOKEN	**Meaning**: The latch manager could not find a granted or pending request associated with the value on the latch token parameter. The latch manager does not release a latch. This return code does not indicate an error if a routine calls Latch_Release to ensure that a latch is released. For example, if an error occurs when a requestor calls the Latch_Obtain service, the requestor's recovery routine might call Latch_Release to ensure that the requested latch is released. If the error prevented the requestor from obtaining the latch, the recovery routine receives this return code. **Action**: If the return code is not expected, validate that the latch token is the same latch token returned to the caller of Latch_Obtain.

Example

See “LATCHX64 - How to call AMODE 64 latch services” on page 1251 for an example of how to call the 64-bit Latch_Release in assembler language.

Chapter 139. ISGQUERY — Global resource serialization query service

Description

The GRS query service routine is given control from the ISGQUERY macro to:

- Search a resource name list (RNL) for a QNAME/RNAME pair.
- Obtain information on resources and requesters of outstanding ENQ requests.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. Any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	For REQINFO=RNLSEARCH, the caller can be unlocked or hold both a local lock (LOCAL or CML) and the CMSEQDQ lock. For REQINFO=QSCAN, the caller must not hold any locks.
Control parameters:	Control parameters must be in the primary address space or for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The control parameters must be in the same key as the caller. For an AMODE 64 caller, the control parameters can reside in virtual storage above the 2G bar. The user-provided answer area that uses the ANSAREA parameter has the same requirements and restrictions as the control parameters.

Programming requirements

The caller must include the ISGYQUAC macro to get a mapping for the answer area.

Note: The ISGYQUAC macro is stabilized as of z/OS R12.

The caller must include the ISGYCON macro to get the values for the return and reason codes.

The caller must include the ISGRNLE macro to get a mapping for the RNLE.

Restrictions

Do not issue ISGQUERY before the GRS address space has been initialized.

There is a restriction on the number of concurrent resource requests in an address space. These include unauthorized ISGENQ, ENQ, RESERVE, and incomplete GQSCAN and ISGQUERY requests. Reason code ISGQUERYRsn_MaxConcurrentRequests is issued if ISGQUERY would cause this limit to be exceeded.

When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=QSCAN must have at least READ authorization to the ISG.QSCANSERVICES.AUTHORIZATION resource in the FACILITY class. When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=LATCHECA must have at least READ authorization to the ISG.LATCHECASERVICES.AUTHORIZATION resource in the FACILITY class. You can activate the multilevel security support through the SETROPTS MLACTIVE option in RACF. For general information about defining profiles in the FACILITY class, see *z/OS Security Server RACF Command Language Reference* and *z/OS Security Server RACF Security Administrator's Guide*. For information about multilevel security, see *z/OS Planning for Multilevel Security and the Common Criteria*.

Callers who specify REQINFO=LATCHECA must not hold any FRRs.

This macro supports multiple versions. Some keywords are unique to certain versions. For more information, see the description of the “,PLISTVER=IMPLIED_VERSION” on page 1314 parameter and the common criteria.

Input register information

Before issuing the ISGQUERY macro, the caller does not have to place any information into any general-purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0.

1
Used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
Return code.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

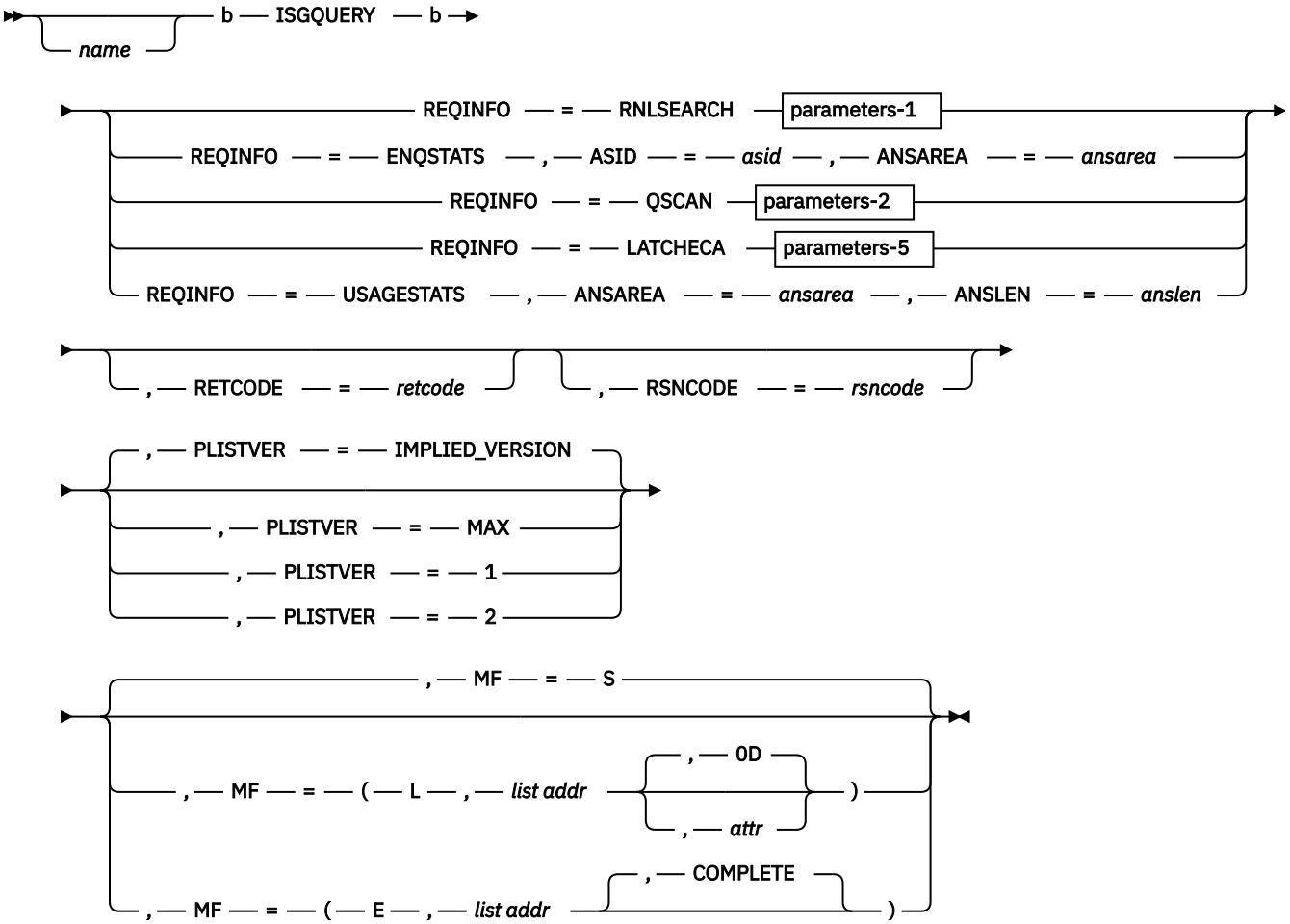
In general, the narrower the search parameters (particularly QNAME and RNAME), the less time the query takes. Using both a specific QNAME and a specific RNAME gives better performance than using patterns.

The use of GATHERFROM=SYSPLEX might greatly degrade the performance of the query request.

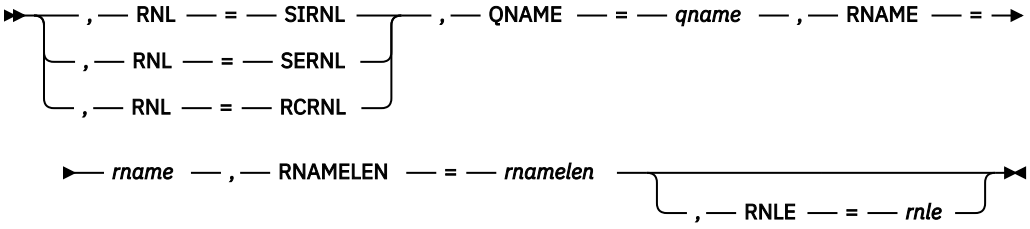
Polling for ENQ contention through GQSCAN or ISGQUERY is not recommended. See the *z/OS MVS Planning: Global Resource Serialization* and *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about monitoring contention through ENF 51.

Syntax

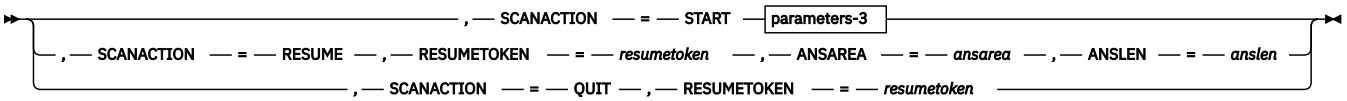
ISGQUERY syntax



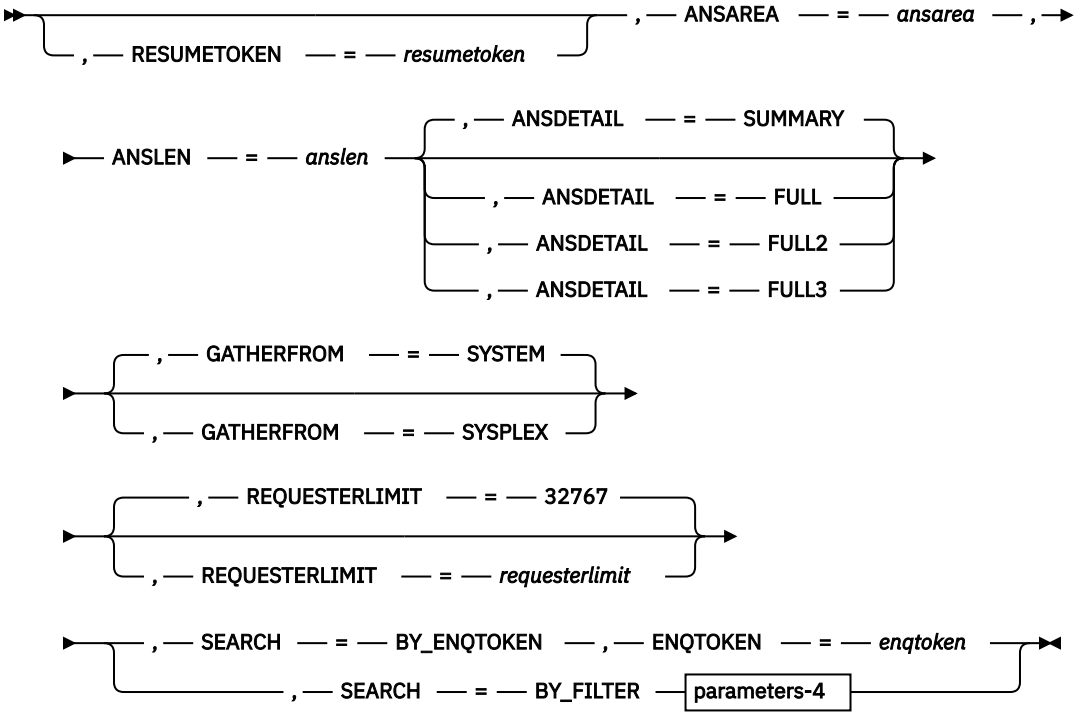
parameters-1



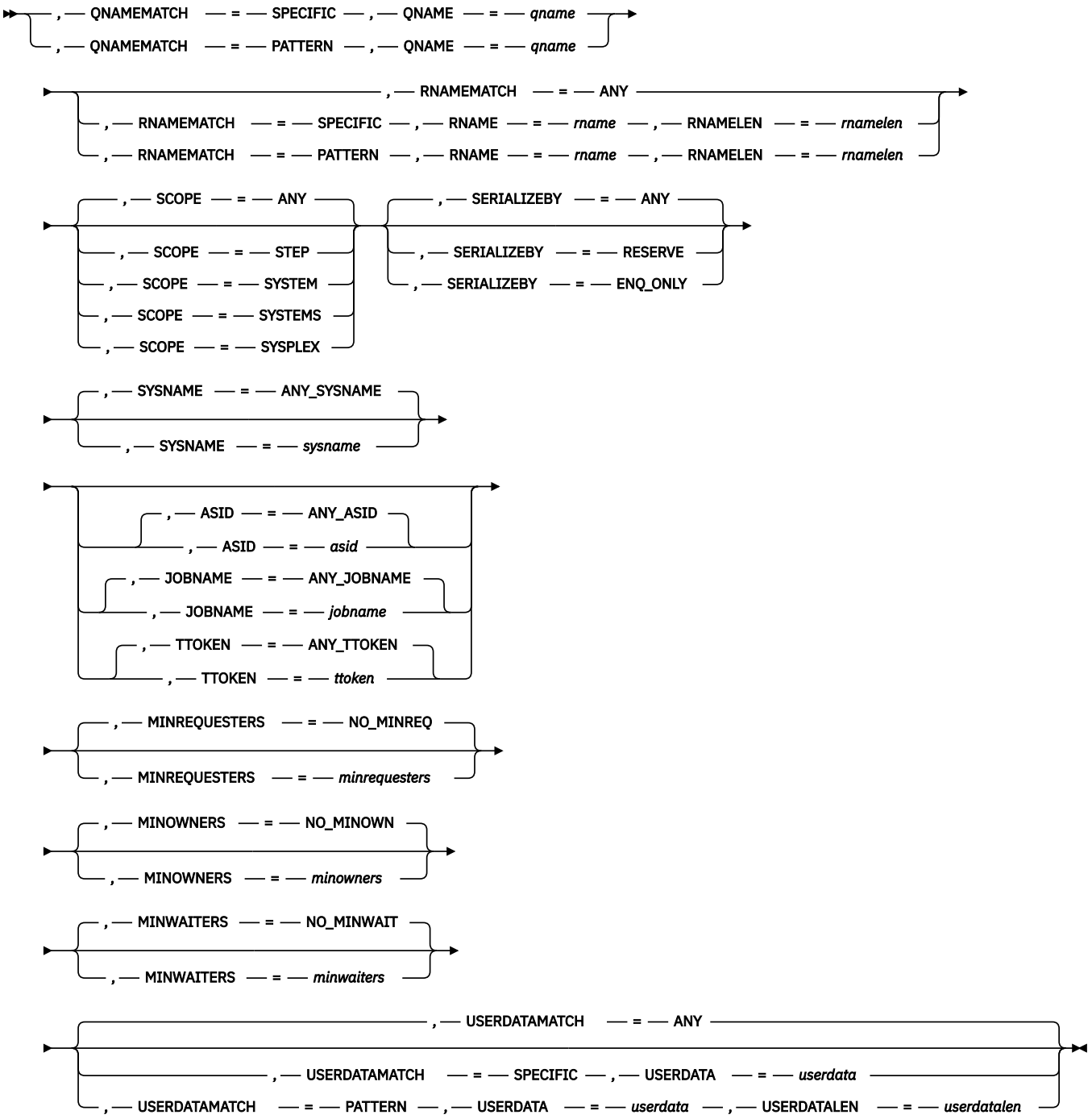
parameters-2



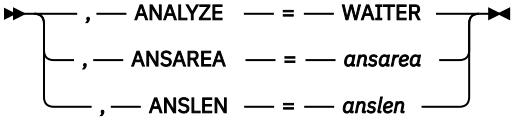
parameters-3



parameters-4



parameters-5



Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1 that is the name on the ISGQUERY macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,ANALYZE=WAITER
When REQINFO=LATCHECA is specified, a required output parameter, which queries LATCHECA waiter data to determine if any long-term latch contention exists that might be cause for concern. ISGQUERY returns only LATCHECA data for waiters.

,ANSAREA=*ansarea*
When REQINFO=ENQSTATS is specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAC. A header area, which is mapped by DSECT ISGYQUAAHdr, is returned followed by additional data, two entries mapped by ISGYQUAASys and two entries mapped by ISGYQUAASp.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSAREA=*ansarea*
When REQINFO=LATCHECA is specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAC. A header area, which is mapped by DSECT ISGYQUAAHdr, is returned followed by additional data mapped by ISGYQUAALd and ISGYQUAALrd.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSAREA=*ansarea*
When REQINFO=USAGESTATS is specified, a required output parameter, which is to contain the returned information. The area is mapped by macro ISGYQUAC. A header area, which is mapped by DSECT ISGYQUAAHdrUs, is returned followed by additional data mapped by ISGYQUAAUs.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSAREA=*ansarea*
When REQINFO=QSCAN is specified, this required output parameter contains the returned information. The area is mapped by macro ISGYQUAC. A header area, which is mapped by DSECT ISGYQUAAHdr, is returned followed by additional data mapped by ISGYQUAARs, ISGYQUAARsx, ISGYQUAARq, and ISGYQUAARqx.

Note: The ANSDETAIL specified determines which data is returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,ANSDETAIL=SUMMARY
,ANSDETAIL=FULL
,ANSDETAIL=FULL2
,ANSDETAIL=FULL3
When SCANACTION=START and REQINFO=QSCAN are specified, an optional parameter that indicates the detail level of the information that should be returned in the answer area. The default is ANSDETAIL=SUMMARY.

,ANSDETAIL=SUMMARY
Indicates to return only ISGYQUAAHdr, ISGYQUAARs, and ISGYQUAARq answer area data records. See ISGYQUAC mapping macro to know what data is returned in each type of record.

,ANSDETAIL=FULL
Indicates to return ISGYQUAAHdr, ISGYQUAARs, ISGYQUAARq, and ISGYQUAARqx answer area data records. See ISGYQUAC mapping macro to know what data is returned in each type of record.

,ANSDETAIL=FULL2
Indicates that in addition to the records returned by ANSDETAIL=FULL, the ISGYQUAARsx, and the larger FULL2 version of the ISGYQUAARqx is returned. See ISGYQUAC mapping macro to know what data is returned in each type of record.

,ANSDETAIL=FULL3
Indicates that in addition to the records returned by ANSDETAIL=FULL2, USERDATA is returned for any records that specified USERDATA on ISGENQ.

Note: When GATHERFROM=SYSPLEX is specified and GRS is operating in STAR mode, USERDATA is not returned for any global requests. See ISGYQUAC mapping macro to know what data is returned in each type of record.

,ANSLEN=*anslen*
When SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the length of the answer area provided. The minimum size is the amount needed to describe a single resource with a single requester. Use an answer area length of at least 4 KB.

- For ANSDETAIL=SUMMARY, the minimum is defined by constant ISGYQUAA_kQSCANMinSummaryAnslen.
- For ANSDETAIL=FULL, the minimum is defined by constant ISGYQUAA_kQSCANMinFullAnslen.
- For ANSDETAIL=FULL2, the minimum is defined by constant ISGYQUAA_kQSCANMinFull2Anslen.
- For ANSDETAIL=FULL3, the minimum is defined by constant ISGYQUAA_kQSCANMinFull3Anslen.

The length of the answer area is at least 4k.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,ANSLEN=*anslen*
When SCANACTION=RESUME and REQINFO=QSCAN are specified, a required input parameter that is the length of the answer area provided. The minimum size is the amount needed to describe a single resource with a single requester. Use an answer area length of at least 4 KB. For ANSDETAIL=SUMMARY, the minimum is defined by constant ISGYQUAA_kQSCANMinSummaryAnslen. For ANSDETAIL=FULL, the minimum is defined by constant ISGYQUAA_kQSCANMinFullAnslen. For ANSDETAIL=FULL2, the minimum is defined by constant ISGYQUAA_kQSCANMinFull2Anslen. For ANSDETAIL=FULL3, the minimum is defined by constant ISGYQUAA_kQSCANMinFull3Anslen. use an answer area length of at least 4 KB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,ANSLEN=*anslen*
When REQINFO=LATCHECA is specified, a required input parameter that is the length of the answer area provided. The minimum size is the amount needed to describe a single resource with a single requester. Use an answer area length of at least 4 K.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,ANSLEN=*anslen*
When REQINFO=USAGESTATS is specified, a required input parameter that is the length of the answer area provided. The minimum size is the amount needed to describe the ENQ, QScan, and latch usage of a single address space and the usage information for terminated address spaces. The minimum is defined by constant ISGYQUAA_kUSAGESTATSMinAnslen. Use an answer area length of at least 4 KB.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,ASID=*asid*
When REQINFO=ENQSTATS is specified, a required input parameter that is the ASID of the address space-specific information to be returned.

Note: ASIDs are reusable. Once an address space has terminated another can be created with the same ASID.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,ASID=*asid*
,ASID=ANY_ASID
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the ASID of the requesting tasks for which resource information is to be returned. Only information on requester with that ASID is returned.

Note: ASID is reusable. Once an address space has terminated another can be created with the same ASID.

The default is ANY_ASID.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,ENQTOKEN=*enqtoken*
When SEARCH=BY_ENQTOKEN, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the ENQToken of the request that is to be queried. Note: ENQTokens are only valid on the system where the ENQ request was made.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,GATHERFROM=SYSTEM
,GATHERFROM=SYSPLEX
When SCANACTION=START and REQINFO=QSCAN are specified, an optional parameter that designates the extent to which the search is taken. Information about other systems is always available locally in a global resource serialization ring complex, so this keyword is ignored and forced to GATHERFROM=SYSTEM.

Use the SYSNAME keyword to obtain only information about one particular system.

Note: Only SYSTEMS scope information is obtained from other systems in the global resource serialization complex.

The default is GATHERFROM=SYSTEM.

,GATHERFROM=SYSTEM
Indicates to search only the caller's system. The answer area data contains information about requester on other systems in the complex only if that information is already available on the caller's system. The returned information might be incomplete regarding requester on other systems, including counts of the number of requester for a resource. If performance is an issue, use GATHERFROM=SYSTEM. This request is always handled without placing the caller's dispatchable unit into a wait.

,GATHERFROM=SYSPLEX
Indicates to search the caller's sysplex. The answer area data contains information about requesters in the entire sysplex. If complete information regarding requesters in the sysplex is required use GATHERFROM=SYSPLEX. There are significant performance implications for this search and the caller might be suspended while the information is being gathered. Do not specify GATHERFROM=SYSPLEX if this condition cannot be tolerated.

GATHERFROM=SYSPLEX is mutually exclusive with the USERDATAMATCH=SPECIFIC and USERDATAMATCH=PATTERN filter options.

When global resource serialization is in STAR mode, GATHERFROM=SYSPLEX with ANSDETAIL=FULL3 results in no user data being returned for global requests.

,JOBNAME=*jobname*
,JOBNAME=ANY_JOBNAME
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the job name of the requesting tasks for which resource information is to be returned. Only information on requesters with that job name is returned. The default is ANY_JOBNAME.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,MINOWNERS=*minowners*
,MINOWNERS=NO_MINOWN
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the minimum number of owners of a resource required for that resource to be returned. If any of the conditions specified by MINREQUESTERS, MINOWNERS, or MINWAITERS is met, even if the other two are not met, information for that resource and its requester is returned. The default is NO_MINOWN.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,MINREQUESTERS=*minrequesters*
,MINREQUESTERS=NO_MINREQ
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the minimum number of owners plus waiters for a resource required for that resource to be returned. If any of the conditions specified by MINREQUESTERS, MINOWNERS, or MINWAITERS is met, even if the other two are not met, information for that resource and its requesters is returned. The default is NO_MINREQ.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,MINWAITERS=*minwaiters*
,MINWAITERS=NO_MINWAIT
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the minimum number of waiters for a resource required for that resource to be returned. If any of the conditions specified by MINREQUESTERS, MINOWNERS, or MINWAITERS is met, even if the other two are not met, information for that resource and its requester is returned. The default is NO_MINWAIT.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field, or specify a literal decimal value.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
,PLISTVER=2

An optional input parameter in the 1-2 range that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, if you use the currently available parameters:
  - ANSAREA
  - ANSDETAIL
  - ANSLEN
  - ASID
  - ENQTOKEN
  - GATHERFROM
  - JOBNAME
  - MINOWNERS
  - MINREQUESTERS
  - MINWAITERS
  - QNAME
  - QNAMEMATCH
  - REQINFO
  - REQUESTERLIMIT
  - RESUMETOKEN
  - RNAME
  - RNAMELEN
  - RNAMEMATCH
  - RNL
  - RNLE
  - SCANACTION
  - SCOPE
  - SEARCH
  - SERIALIZEBY
  - SYSNAME
  - TTOKEN
- **2**, which supports both the following parameters and those from version 1:
  - USERDATA

- USERDATALEN
- USERDATAMATCH

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1, or 2

,QNAME=*qname*
When REQINFO=RNLSEARCH is specified, a required input parameter that is the Qname of the resource for which the RNLs are to be searched.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,QNAME=*qname*
When QNAMEMATCH=SPECIFIC, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the specific Qname of the resources to be returned.

To code:To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,QNAME=*qname*
When QNAMEMATCH=PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is a pattern Qname to match the resources to be returned.

The Qname pattern is 8 characters where "?" matches any single character, and "*" matches any string of zero or more characters.

Note: All trailing blanks are ignored when matching Qnames to Qname patterns.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,QNAMEMATCH=SPECIFIC
,QNAMEMATCH=PATTERN
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required parameter.

,QNAMEMATCH=SPECIFIC
Indicates to only return information on resources that exactly match the specified specific Qname.

,QNAMEMATCH=PATTERN
Indicates to only return information on resources that match the specified Qname pattern.

REQINFO=RNLSEARCH
REQINFO=ENQSTATS
REQINFO=QSCAN
REQINFO=LATCHECA
REQINFO=USAGESTATS
A required parameter that designates the data to be returned.

REQINFO=RNLSEARCH
Indicates to search a specific RNL for a resource name.

Therefore, the CMSEQDQ lock serializes the use of the RNLs, so holding this lock ensures that the RNL does not change and the returned RNLE is valid on the current RNLs.

During an RNL change, the currently active RNLs are searched.

For more information about how a resource can be changed by the system, see the TEST=YES function in Chapter 126, “ISGENQ — Global resource serialization ENQ service,” on page 1205.

REQINFO=ENQSTATS
Indicates to return information related to ENQ counts.

REQINFO=QSCAN
Indicates to search the global resource serialization queues for resource and requester information.

REQINFO=LATCHECA
Indicates to search the global resource serialization queues for query latch enhanced contention analysis (ECA) data for waiters that might indicate contention issues.

Note: The LATCHECA search does not return data for blockers or dependency data.

REQINFO=USAGESTATS
Indicates to search the global resource serialization queues for address space level contention information related to ENQs (all scopes) and latches (all latch sets). Global resource serialization gathers latch statistics in requester and latch set owner address space categories. The statistics are provided for all address spaces as follows:

- ENQ by scope: this includes contention counts, total delay times, and the sum of the squared delay (SUMSQ) times. The SUMSQ times can be used to compute the standard deviation.
- Latch: For both requester and latch set owners, this includes contention counts, total delay times, and the sum of the squared delay (SUMSQ) times
- ENQ usage counts.

Note: Latch counts are kept in "fast counts" in latch sets and not on an address space basis.

,REQUESTERLIMIT=*requesterlimit*
,REQUESTERLIMIT=32767
When SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the maximum number of requesters (owners and waiters) to be returned for each individual resource. Only resource-related information is returned if 0 is specified. The value range of REQUESTERLIMIT is 0 to 2¬15-1 (32767). The default is 32767.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,RESUMETOKEN=*resumetoken*
When SCANACTION=START and REQINFO=QSCAN are specified, an optional output parameter that is the resume token for this search. When RESUMETOKEN is specified, a reason code of ISGQUERYRsn_AnswerAreaFull indicates that the token can be used to resume the scan on a subsequent call. Subsequently, if the return code indicates that the search can be resumed, a SCANACTION=RESUME or SCANACTION=QUIT with the returned resume token must be issued.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,RESUMETOKEN=*resumetoken*
When SCANACTION=RESUME and REQINFO=QSCAN are specified, a required input/output parameter that is the resume token from a previously started search. If the search does not complete the resume token can be used to resume the search on a subsequent call. Check the return code to determine if the resume token can be used to resume the scan. Subsequently, if the return code indicates that the search can be resumed, a SCANACTION=RESUME or SCANACTION=QUIT with the returned resume token must be issued.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,RESUMETOKEN=*resumetoken*
When SCANACTION=QUIT and REQINFO=QSCAN are specified, a required input/output parameter that is the resume token from a previously started search. Any global resource serialization storage associated with the search is freed, and the resume token is cleared to binary zeros.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RNAME=*rname*
When REQINFO=RNLSEARCH is specified, a required input parameter that is the RName of the resource for which the RNLs are to be searched.

The RName pattern is a string of characters where "?" matches any single character, and "*"matches any string of zero or more characters.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RNAME=*rname*
When RNAMEMATCH=SPECIFIC, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the specific RName of the resources to be returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RNAME=*rname*
When RNAMEMATCH=PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is a pattern RName to match the resources to be returned. The RName pattern is a string of characters where '?' matches any single character, and '*' matches any string of zero or more characters.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RNAMELEN=*rnamelen*
When REQINFO=RNLSEARCH is specified, a required input parameter that is the length of the RName. The specified length can be 1 - 255.

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

,RNAMELEN=*rnamelen*
When RNAMEMATCH=SPECIFIC, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the length of the RName. The specified length can be 1 - 255.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-byte field.

,RNAMELEN=*rnamelen*
When RNAMEMATCH=PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the length of the RName pattern. The specified length can be 1 - 255.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-byte field.

,RNAMEMATCH=ANY
,RNAMEMATCH=SPECIFIC
,RNAMEMATCH=PATTERN
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required parameter.

,RNAMEMATCH=ANY
Indicates to return information on resources with any RName.

,RNAMEMATCH=SPECIFIC
Indicates to return information on resources that exactly match the specified specific RName.

,RNAMEMATCH=PATTERN
Indicates to return information on resources that match the specified RName pattern.

,RNL=SIRNL
,RNL=SERNL
,RNL=RCRNL
When REQINFO=RNLSEARCH is specified, a required parameter that indicates which resource name list (RNL) is to be searched.

,RNL=SIRNL
Indicates to search the system inclusion RNL.

,RNL=SERNL
Indicates to search the systems exclusion RNL.

,RNL=RCRNL
Indicates to search the reserve conversion RNL.

,RNLE=*rnle*
When REQINFO=RNLSEARCH is specified, an optional output parameter that is a copy of the matching RNLE. The caller must include the ISGRNLE macro to get a mapping for the RNLE.

Note: The RNLE returned depends on the version of the parameter list. If a new version of the RNLE should be introduced, it might require a larger character field. Explicitly state the PLISTVER to ensure that the size of the RNLE returned does not change.

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value is left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,SCANACTION=START
,SCANACTION=RESUME
,SCANACTION=QUIT
When REQINFO=QSCAN is specified, a required parameter that designates whether to start, resume, or quit a QScan.

,SCANACTION=START
Indicates to start a search of the global resource serialization queues.

,SCANACTION=RESUME
Indicates to resume a previously started search.

,SCANACTION=QUIT
Indicates to quit a previously started search. If a started search has not completed, it must be either resumed until it completes or ended with SCANACTION=QUIT.

,SCOPE=ANY
,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
,SCOPE=SYSPLEX
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional parameter that is the scope of the resources to be returned.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller's system.

The default is SCOPE=ANY.

,SCOPE=ANY
Indicates to return information on resources with any scope.

,SCOPE=STEP
Indicates to return information on resources with a scope of STEP.

,SCOPE=SYSTEM
Indicates to return information on resources with a scope of SYSTEM.

,SCOPE=SYSTEMS
Indicates to return information on resources with a scope of SYSTEMS or SYSPLEX.

,SCOPE=SYSPLEX
Indicates to return information on resources with a scope of SYSTEMS or SYSPLEX. (SYSPLEX is an alias for SYSTEMS.)

,SEARCH=BY_ENQTOKEN
,SEARCH=BY_FILTER
When SCANACTION=START and REQINFO=QSCAN are specified, a required parameter that designates the method to search for resources.

,SEARCH=BY_ENQTOKEN
Indicates to search using a specific ENQToken. Information is returned about the requester of the ENQ and the resource for which the ENQ was requested.

,SEARCH=BY_FILTER
Indicates to search on resource and requester characteristics using filters. Information is returned about the resources and requesters that match the search criteria.

,SERIALIZEBY=ANY
,SERIALIZEBY=RESERVE
,SERIALIZEBY=ENQ_ONLY
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional parameter that indicates if information should be returned depending on whether the requests are serialized by device reserves. The default is SERIALIZEBY=ANY.

,SERIALIZEBY=ANY
Indicates to return information on requests of any type.

,SERIALIZEBY=RESERVE
Indicates to return information on reserve requests that were not converted.

,SERIALIZEBY=ENQ_ONLY
Indicates to return information on requests that do not result in a device reserve. This includes reserve requests that were converted to global ENQs. Answer area bit ISGYQUAARqReserveConverted is set for reserve requests that were converted.

,SYSNAME=*sysname*
,SYSNAME=ANY_SYSNAME
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the system name of the requesting tasks for which resource information is to be returned. Only information on requester in that system is returned. If GATHERFROM=SYSTEM is specified (or is the default), SYSNAME might only be the name of the caller's system or the default of ANY_SYSNAME.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller's system.

The default is ANY_SYSNAME.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,TTOKEN=*ttoken*
,TTOKEN=ANY_TTOKEN
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional input parameter that is the task token of the requesting task for which resource information is to be returned. Only information on that requester is returned. The TToken specified is valid only on the current system.

Note: The TToken of requesters is unavailable for ENQs obtained before the global resource serialization address space was created. The TToken filter will not match those ENQ requesters.

The default is ANY_TTOKEN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,USERDATA=*userdata*
When USERDATAMATCH=SPECIFIC, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the specific UserData of the requests to be returned.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,USERDATA=*userdata*
When USERDATAMATCH=PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is a pattern UserData to match the requests to be returned. The UserData pattern is a string of characters where '?' matches any single character, and '*' matches any string of zero or more characters.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,USERDATALEN=*userdatalen*
When USERDATAMATCH=PATTERN, SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, a required input parameter that is the length of the given UserData pattern. The specified length can be 1 - 32.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value.

,USERDATAMATCH=ANY
,USERDATAMATCH=SPECIFIC
,USERDATAMATCH=PATTERN
When SEARCH=BY_FILTER, SCANACTION=START and REQINFO=QSCAN are specified, an optional parameter that indicates which requests to return. The default is USERDATAMATCH=ANY.

,USERDATAMATCH=ANY
Indicates to return information on request with any USERDATA, including those with no USERDATA.

,USERDATAMATCH=SPECIFIC
Indicates to only return requests that have USERDATA that exactly matches the specified USERDATA. For information about specifying USERDATA on an ISGENQ request, see Chapter 126, “ISGENQ — Global resource serialization ENQ service,” on page 1205. Note that USERDATA can only be attached to a request through the ISGENQ interface.

This request requires a version 2 parameter list.

GATHERFROM=SYSPLEX is mutually exclusive with the USERDATAMATCH=SPECIFIC option.

,USERDATAMATCH=PATTERN
Indicates to only return information on requests that match the specified UserData pattern. For information about specifying USERDATA on an ISGENQ request, see Chapter 126, “ISGENQ — Global resource serialization ENQ service,” on page 1205.

All trailing blanks are not ignored when matching USERDATA to USERDATA patterns. For example, if the USERDATA is ABC123, and the pattern used to search is A*3, it does not match. A pattern such as A*3* does match.

Note: Userdata can only be attached to a request through the ISGENQ interface.

This request requires a version 2 parameter list.

GATHERFROM=SYSPLEX is mutually exclusive with the USERDATAMATCH=PATTERN option.

ABEND codes

None.

Return and reason codes

When the ISGQUERY macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro ISGYCON provides equate symbols for the return and reason codes.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel can request the entire reason code, including the **xxxx** value.

Table 125. Return and Reason Codes for the ISGQUERY Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
00	—	**Equate Symbol**: ISGQUERYRc_OK **Meaning**: ISGQUERY request successful. For REQINFO=RNLSEARCH, a matching RNLE was found for the given resource name. For REQINFO=QSCAN, processing complete and data has been copied into the answer area. There are no more data to return. **Action**: None required.
04	—	**Equate Symbol**: ISGQUERYRc_Warn **Meaning**: Warning. ISGQUERY completed successfully, however a warning has been issued. **Action**: Refer to action under the individual reason code.
04	xxxx0401	**Equate Symbol**: ISGQUERYRsn_NoMatchingRNLE **Meaning**: For a REQINFO=RNLSEARCH request. No matching RNLE was found for the given resource name. **Action**: No action required.
04	xxxx0402	**Equate Symbol**: ISGQUERYRsn_RNLChangeInProgress **Meaning**: For a REQINFO=RNLSEARCH request. A matching RNLE was found for the given resource name, but an RNL change is in progress in the system. **Action**: No action required.
04	xxxx0403	**Equate Symbol**: ISGQUERYRsn_GRSRNLExclude **Meaning**: For a REQINFO=RNLSEARCH request. GRSRNL=EXCLUDE is in effect. When GRSRNL=EXCLUDE the RNLs are not used and all SYSTEMS scope requests are forced to SYSTEM. An alternative serialization product can be in use. No RNLE is returned. **Action**: No action required.
04	xxxx0404	**Equate Symbol**: ISGQUERYRsn_NoMatchingResources **Meaning**: For REQINFO=QSCAN and REQINFO=LatchECA requests. While scanning the queues, no resources were found that match the caller's request. **Action**: No action required.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
04	xxxx0405	**Equate Symbol**: ISGQUERYRsn_AnswerAreaFull **Meaning**: For a REQINFO=QSCAN request. ISGQUERY has provided some data, however the answer area is too small to contain all the requested data. **Action**: The user should process the data in the answer area. If RESUMETOKEN was not specified on the request and more information is needed, re-issue the request with a larger answer area or specify a resume token. If RESUMETOKEN was specified, either issue a REQINFO=QSCAN SCANACTION=RESUME request with the returned resume token to continue continue the scan, or issue REQINFO=QSCAN SCANACTION=QUIT to end the search.
04	xxxx0406	**Equate Symbol**: ISGQUERYRsn_GRSNone **Meaning**: For a REQINFO=RNLSEARCH request. GRS=NONE is in effect. When GRS=NONE the RNLs are not used and all requests are serialized only within the current system. Note that though both scope SYSTEM and SYSTEMS requests are local to the current system, they still represent separate resouces and are NOT serialized with each other.
08	—	**Equate Symbol**: ISGQUERYRc_ParmError **Meaning**: ISGQUERY request specified parameters in error. **Action**: Refer to action under the individual reason code.
08	xxxx0801	**Equate Symbol**: ISGQUERYRsn_BadPlistAddress **Meaning**: Unable to access parameter list. **Action**: Check that the entire parameter list is addressable. If in AR-mode, check that the ALET of the parameter list is correct. Note that if this macro is issued in AR-mode, SYSSTATE ASCENV=AR must be issued before this macro. Ensure that the storage is in the same key as the caller.
08	xxxx0802	**Equate Symbol**: ISGQUERYRsn_BadPlistALET **Meaning**: Bad parameter list ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the parameter list is valid. Its access register might have been set up properly.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0803	**Equate Symbol**: ISGQUERYRsn_BadPlistVersion **Meaning**: Bad parameter list version number. The service level of GRS on which the caller is running does not support this version of the ISGQUERY service, or the ISGQUERY parameter list version is lower than the minimum required for parameters that were specified. **Action**: Check that the request has the correct version number. Check for possible storage overlay of the parameter list.
08	xxxx0804	**Equate Symbol**: ISGQUERYRsn_ReservedFieldNotNull **Meaning**: A reserved field in the parameter list is non-zero. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0805	**Equate Symbol**: ISGQUERYRsn_BadReqInfo **Meaning**: Bad REQINFO parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0806	**Equate Symbol**: ISGQUERYRsn_BadRNL **Meaning**: Bad RNL parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0807	**Equate Symbol**: ISGQUERYRsn_BadRNameAddress **Meaning**: Unable to access the RName. **Action**: Ensure that the entire RName field is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Check that specified RName length is correct. Ensure that the storage is in the same key as the caller.
08	xxxx0808	**Equate Symbol**: ISGQUERYRsn_BadRNameALET **Meaning**: Bad RName ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the RName is valid. Its access register might have been set up properly.
08	xxxx0809	**Equate Symbol**: ISGQUERYRsn_BadRNameLen **Meaning**: The RName length specified is not valid. **Action**: Ensure the RName length field contains a number from 1-255.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx080A	**Equate Symbol**: ISGQUERYRsn_BadRNLEAddress **Meaning**: Unable to access RNLE output field. **Action**: Ensure that the entire RNLE field is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Check that RNLE length is correct. Ensure that the storage is in the same key as the caller.
08	xxxx080B	**Equate Symbol**: ISGQUERYRsn_BadRNLEALET **Meaning**: Bad RNLE ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the RNLE is valid. Its access register might have been set up properly.
08	xxxx080C	**Equate Symbol**: ISGQUERYRsn_MutuallyExclusive **Meaning**: Mutually exclusive keywords were specified. **Action**: Check for a possible storage overlay of the parameter list.
08	xxxx080D	**Equate Symbol**: ISGQUERYRsn_BadAnsAreaAddress **Meaning**: Unable to access the answer area. **Action**: Ensure that the entire answer area is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Check that the specified answer area length is correct. Ensure that the storage is in the same key as the caller.
08	xxxx080E	**Equate Symbol**: ISGQUERYRsn_BadAnsAreaALET **Meaning**: Bad answer area ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the answer area is valid. Its access register might have been set up properly.
08	xxxx080F	**Equate Symbol**: ISGQUERYRsn_BadScanAction **Meaning**: Bad SCANACTION parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0810	**Equate Symbol**: ISGQUERYRsn_BadResumeTokenAddress **Meaning**: Unable to access the ResumeToken. **Action**: Ensure that the entire ResumeToken is addressable. If in AR-mode, this field is accessed through its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0811	**Equate Symbol**: ISGQUERYRsn_BadResumeTokenALET **Meaning**: Bad ResumeToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the ResumeToken is valid. Its access register might not have been set up properly.
08	xxxx0812	**Equate Symbol**: ISGQUERYRsn_BadGatherFrom **Meaning**: Bad GATHERFROM parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0813	**Equate Symbol**: ISGQUERYRsn_BadSearch **Meaning**: Bad SEARCH keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0814	**Equate Symbol**: ISGQUERYRsn_BadENQTokenAddress **Meaning**: Unable to access the ENQToken. **Action**: Ensure that the entire ENQToken is addressable. If in AR-mode, this field is accessed via its address and ALET, check that both these values are correct. Ensure that the storage is in the same key as the caller.
08	xxxx0815	**Equate Symbol**: ISGQUERYRsn_BadENQTokenALET **Meaning**: Bad ENQToken ALET. The ALET is neither zero nor is it associated with a valid public entry on the caller's dispatchable unit access list (DU-AL), nor a valid entry for a common area data space. **Action**: Ensure that the ALET of the ENQToken is valid. Its access register might have been set up properly.
08	xxxx0816	**Equate Symbol**: ISGQUERYRsn_BadQNameMatch **Meaning**: Bad QNAMEMATCH keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0817	**Equate Symbol**: ISGQUERYRsn_BadRNameMatch **Meaning**: Bad RNAMEMATCH keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0818	**Equate Symbol**: ISGQUERYRsn_BadScope **Meaning**: Bad SCOPE keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0819	**Equate Symbol**: ISGQUERYRsn_BadSerializeBy **Meaning**: Bad SERIALIZEBY keyword parameter. **Action**: Check for possible storage overlay of the parameter list.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx081A	**Equate Symbol**: ISGQUERYRsn_AnsLenTooSmall **Meaning**: The size of the answer area is not large enough to contain the minimal amount of information. **Action**: Increase the answer area size to at least the minimum required for the specified request. See the provided constants. However, the answer area length should be at least 4k.
08	xxxx081B	**Equate Symbol**: ISGQUERYRsn_ResumeTokenNotValid **Meaning**: The specified resume token is not a valid resume token. **Action**: Ensure the resume token is from a previously started search on the current system.
08	xxxx081C	**Equate Symbol**: ISGQUERYRsn_ResumeTokenTooOld **Meaning**: The specified resume token is from an old search request that has expired. **Action**: Restart the search if more information is needed.
08	xxxx081D	**Equate Symbol**: ISGQUERYRsn_ENQTokenNotValid **Meaning**: The ENQToken specified is not a valid ENQToken. **Action**: Ensure the ENQToken is from a previous ISGENQ request on the current system.
08	xxxx081E	**Equate Symbol**: ISGQUERYRsn_BadRequesterLimit **Meaning**: The REQUESTERLIMIT value specified is not valid. RequesterLimit must be 0 - 2?5-1 (32767). **Action**: Ensure that the requester limit is in the correct range.
08	xxxx081F	**Equate Symbol**: ISGQUERYRsn_NoPossibleMatch **Meaning**: For a REQINFO=QSCAN request. Conflicting parameters were specified such that no resources could possibly match the request. A SYSNAME other than the current system was specified along with SCOPE=STEP, SCOPE=SYSTEM, TTOKEN, or GATHERFROM=SYSTEM. Or SERIALIZEBY=RESERVE was specified with SCOPE=STEP. **Action**: Avoid specifying conflicting parameters.
08	xxxx0820	**Equate Symbol**: ISGQUERYRsn_BadAnsDetail **Meaning**: Bad ANSDETAIL keyword parameter. **Action**: Check for possible storage overlay of the parameter list.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0821	**Equate Symbol**: ISGQUERYRsn_NotAuthToQscan **Meaning**: SETROPTS MLACTIVE is in effect, and the program is not authorized to issue ISGQUERY REQINFO=QSCAN. **Action**: Ensure that the program is running authorized, or is associated with a user id with at least READ access to the best fit FACILITY class resource profile of the form ISG.QSCANSERVICES.AUTHORIZATION and that the FACILITY class is SETROPTS RACLISTed.
08	xxxx0822	**Equate Symbol**: ISGQUERYRsn_BadASID **Meaning**: Bad ASID keyword parameter. **Action**: Ensure that the ASID is valid.
08	xxxx0823	**Equate Symbol**: ISGQUERYRsn_BadUserDataAddress **Meaning**: Unable to access the user data. **Action**: Ensure that the entire USERDATA is addressable. If in AR-mode, this field is accessed by its address and ALET, check that both these values are correct. If this is a USERDATA pattern request, check that specified USERDATA length is correct. Ensure that the storage is in the same key as the caller.
08	xxxx0824	**Equate Symbol**: ISGQUERYRsn_BadUserDataAlet **Meaning**: Bad USERDATA ALET. The ALET is not zero or is it associated with a valid public entry on the caller's Dispatchable Unit Access List (DU-AL), or a valid entry for a common area data space. **Action**: Ensure that the ALET of the USERDATA is valid. Its access register might have been set up properly.
08	xxxx0825	**Equate Symbol**: ISGQUERYRsn_BadUserDataLen **Meaning**: The USERDATA length specified is not valid. **Action**: Ensure the USERDATA length field contains a number in the range 1-32.
08	xxxx0826	**Equate Symbol**: ISGQUERYRsn_BadUserDataMatch **Meaning**: Bad USERDATAMATCH keyword parameter. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0827	**Equate Symbol**: ISGQUERYRsn_BadAnalyze **Meaning**: The ANALYZE keyword parameter is not valid. **Action**: Check for possible storage overlay of the parameter list.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0828	**Equate Symbol**: ISGQUERYRsn_NotAuthToLatchECA **Meaning**: SETROPTS MLACTIVE is in effect and the program is not authorized to issue ISGQUERY REQINFO=LATCHECA. **Action**: Ensure the program is running authorized or is associated with a userid with at least READ access to the best fit FACILITY class resource profile of the form ISG.LATCHECASERVICES.AUTHORIZATION and that the FACILITY class is SETROPTS RACLISTed.
0C	—	**Equate Symbol**: ISGQUERYRc_EnvError **Meaning**: ISGQUERY request has an environment error. **Action**: Refer to action under the individual reason code.
0C	xxxx0C01	**Equate Symbol**: ISGQUERYRsn_SrbMode **Meaning**: ISGQUERY can not be used in SRB mode. **Action**: Avoid using ISGQUERY in SRB mode.
0C	xxxx0C02	**Equate Symbol**: ISGQUERYRsn_NotEnabled **Meaning**: ISGQUERY can not be used disabled. **Action**: Avoid using ISGQUERY when not enabled.
0C	xxxx0C03	**Equate Symbol**: ISGQUERYRsn_ComplexMigrating **Meaning**: For a REQINFO=QSCAN request. The ISGQUERY service failed because the GRS complex was migrating from a ring to a star configuration. **Action**: Retry the request on or more times.
0C	xxxx0C04	**Equate Symbol**: ISGQUERYRsn_CannotObtainLocks **Meaning**: For REQINFO=RNLSEARCH, the local and CMSEQDQ locks could not be obtained. **Action**: Only use ISGQUERY REQINFO=RNLSEARCH when either no locks are held, or both a local lock and the CMSEQDQ lock are held with no other locks.
0C	xxxx0C05	**Equate Symbol**: ISGQUERYRsn_LockHeld **Meaning**: An incorrect lock was held upon entry. For REQINFO=QSCAN, no locks can be held. For REQINFO=RNLSEARCH, either no locks or both a local lock (LOCAL or CML) and the CMDEQDQ lock must be held. **Action**: Avoid using ISGQUERY REQINFO=QSCAN when locks are held. Avoid using ISGQUERY REQINFO=RNLSEARCH when locks other than both a local lock and the CMSEQDQ lock are held.

Table 125. Return and Reason Codes for the ISGQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
0C	xxxx0C06	**Equate Symbol**: ISGQUERYRsn_MaxConcurrentRequests **Meaning**: For a REQINFO=QSCAN request. The answer area was filled before queue scan processing completed, and reason code ISGQUERYRsn_AnswerAreaFull would have been issued. However, RESUMETOKEN was specified, but the limit for the number of concurrent resource requests (ISGENQ, ENQ, RESERVE, GQSCAN, and ISGQUERY) has been reached. The data in the answer area is valid, but incomplete. The scan cannot be resumed. **Action**: Retry the request one or more times. If the problem persists, consult your system programmer. For more information on concurrent count limits and how the system can be tuned when necessary, see *z/OS MVS Planning: Global Resource Serialization*.
0C	xxxx0C07	**Equate Symbol**: ISGQUERYRsn_RingResumeInStar **Meaning**: For a REQINFO=QSCAN request. The caller attempted to resume a scan that was started when the global resource serialization complex, which is now in star mode, was in ring mode. **Action**: Reissue the original request.
0C	xxxx0C08	**Equate Symbol**: ISGQUERYRsn_InsufficientStorage **Meaning**: For a REQINFO=QSCAN request. The ISGQUERY service could not obtain storage to satisfy the request. **Action**: Retry the request one or more times.
0C	xxxx0C09	**Equate Symbol**: ISGQUERYRsn_FRRHeld, **Meaning**: For a REQINFO=LATCHECA request. The caller issued ISGQUERY with a functional recover routine (FRR) established. **Action**: Avoid issuing ISGQUERY REQINFO=LATCHECA when using functional recovery routines.
10	—	**Equate Symbol**: ISGQUERYRc_CompError **Meaning**: Component Error **Action**: Contact the IBM Support Center. The reason code contains internal diagnostic information.

Examples

Use these examples as a guide.

```
* ********************************************************************
*   Search the Systems Inclusion RNL for a resource name
* ********************************************************************

         ISGQUERY REQINFO=RNLSEARCH,RNL=SIRNL,                         X
               QNAME=MYQNAME,RNAME=MYRNAME,RNAMELEN=MYRNAMELEN,        X
               RETCODE=MYRC,RSNCODE=MYRSN

* ********************************************************************
*  Query information on a request specified by ENQToken
```

```
* ******************************************************************

         ISGQUERY REQINFO=QSCAN,SCANACTION=START,                      X
               ANSAREA=MYAREA,ANSLEN=MYAREALEN,                        X
               SEARCH=BY_ENQTOKEN,ENQTOKEN=MYENQTOKEN,                 X
               RETCODE=MYRC,RSNCODE=MYRSN

* ******************************************************************
*  Start a resumable query for resources of a specific job that
*  matches a specific QNAME and pattern RNAME
* ******************************************************************

         ISGQUERY REQINFO=QSCAN,SCANACTION=START,                      X
               ANSAREA=MYAREA,ANSLEN=MYAREALEN,                        X
               SEARCH=BY_FILTER,QNAMEMATCH=SPECIFIC,QNAME=MYQNAME,     X
               RNAMEMATCH=PATTERN,RNAME==CL7'ABC?23*',RNAMELEN=7,      X
               USERDATAMATCH=SPECIFIC,USERDATA=MYUDATA,                X
               JOBNAME=MYJOBNAME,RESUMETOKEN=MYRESTOKEN,RETCODE=MYRC,  X
               RSNCODE=MYRSN

* ******************************************************************
*  Start a resumable query for resources of a specific job that
*  matches a specific QNAME and pattern RNAME
* ******************************************************************

ISGQUERY REQINFO=QSCAN,SCANACTION=START,                              X
         ANSAREA=MYAREA,ANSLEN=MYAREALEN,                            X
         SEARCH=BY_FILTER,QNAMEMATCH=SPECIFIC,QNAME=MYQNAME,         X
         RNAMEMATCH=PATTERN,RNAME==CL7'ABC?23*',RNAMELEN=7,          X
         USERDATAMATCH=PATTERN,USERDATA=MYUDATAP,USERDATALEN=7,      X
         JOBNAME=MYJOBNAME,RESUMETOKEN=MYRESTOKEN,RETCODE=MYRC,      X
         RSNCODE=MYRSN

MYUDATA  DC    CL32'MY USERDATA'
MYUDATAP DC    CL7'M??USE*'

* ******************************************************************
*  Resume a query that was started but not completed
* ******************************************************************

         ISGQUERY REQINFO=QSCAN,SCANACTION=RESUME,                     X
               RESUMETOKEN=MYRESTOKEN,                                 X
               ANSAREA=MYAREA,ANSLEN=MYAREALEN,                        X
               RETCODE=MYRC,RSNCODE=MYRSN

* ******************************************************************
*  Quit a query that was started but not completed
* ******************************************************************

         ISGQUERY REQINFO=QSCAN,SCANACTION=QUIT,                       X
               RESUMETOKEN=MYRESTOKEN,                                 X
               RETCODE=MYRC,RSNCODE=MYRSN
```

```
* ******************************************************************
*  Gather ENQ statistics for a particular address space
* ******************************************************************

          ISGQUERY REQINFO=ENQSTATS,                                   X
                ANSAREA=MYAREA,ASID=MYASID,                            X
                RETCODE=MYRC,RSNCODE=MYRSN
```

```
* ******************************************************************
*  Gather query latch enhanced contention analysis (LATCHECA) data from the  *
*  global resource serialization queues for waiters delayed because of  *
*  contention                                                          *
********************************************************************

         ISGQUERY REQINFO=LATCHECA,ANALYZE=WAITER,ANSAREA=MYAREA,      X
               ANSLEN=MYAREALEN,RETCODE=MYRC,RSNCODE=MYRSN             X
```

```
* ******************************************************************
*  Gather  address space level contention information related to ENQs   *
*  (all scopes) and latches (all latch sets) from the                  *
*  global resource serialization queues                                *
********************************************************************
```

```
          ISGQUERY REQINFO=USAGESTATS,ANSAREA=MYAREA,ANSLEN=MYAREALEN,  X
                RETCODE=MYRC,RSNCODE=MYRSN                              X
```

For more information on global resource serialization, see *z/OS MVS Planning: Global Resource Serialization*.

Chapter 140. ITTFMTB — Generate component trace format table

Description

ITTFMTB generates a table called the component trace format table. It can also generate a map of the table. IPCS uses this table to control the formatting of trace data for program events that occur when the system runs. When you use ITTFMTB to generate information in the table, you are specifying the formatting style of the trace data. For information about IPCS, see *z/OS MVS IPCS User's Guide* and *z/OS MVS IPCS Customization*.

Invoke the macro once to define the beginning of the table and once to define the end of the table. In between, you can invoke the macro repeatedly to define the individual formats for the various traceable events.

This macro generates nonexecutable code, and therefore is not sensitive to the execution environment.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held

Programming requirements

None.

Restrictions

None.

Register information

This macro does not use any registers.

Performance implications

None.

Syntax

The ITTFMTB macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ITTFMTB.
ITTFMTB	
ƀ	One or more blanks must follow ITTFMTB.
MAP	Required choice. Select one of four options.
TABLEDATA=*tabname*	*tabname*: Symbol up to eight characters long.
EVENTDATA=*eventid*	*eventid*: A-type address.
TABLEEND	
,ENTRYLENGTH=*elength*	Optional with TABLEDATA and not otherwise allowed.
	elength: A-type address.
,LOCBUFNAME=*bufname*	Required choice with TABLEDATA and not otherwise allowed.
	bufname: Symbol up to eight characters long.
,LOCBUFADDR=*bufaddr*	*bufaddr*: A-type address.
,FILTERNAME=*pgmname*	Optional choice with TABLEDATA and not otherwise allowed.
	pgmname: Symbol up to eight characters long.
,FILTERADDR=*pgmaddr*	*pgmaddr*: A-type address.
,MNEMONIC=*mnemonic*	Required with EVENTDATA and not otherwise allowed.
	mnemonic: Symbol up to 32 characters long.
,DESCRIPTION=*text*	Required with EVENTDATA and not otherwise allowed. *text*: Symbol up to 32 characters long.
,MODELNAME=*modelname*	Optional choice with EVENTDATA and not otherwise allowed.
	modelname: Symbol up to eight characters long.
,MODELADDR=*modeladdr*	,MODELADDR=*modeladdr*
,FORMATNAME=*pgmname*	Optional choice with EVENTDATA and not otherwise allowed.

Syntax	Description
	pgmname: Symbol up to eight characters long.
,FORMATADDR=*pgmaddr*	*pgmaddr*: A-type address.
,OFFSETASID=*(ids)*	Optional with EVENTDATA and not otherwise allowed.
	ids: One or more A-type addresses, separated by commas.
,OFFSETJOBNAME=*(offsets)*	Optional with EVENTDATA and not otherwise allowed.
	offsets: One or more A-type addresses, separated by commas.
,VIEWSUMMARY=*scode*	Optional with EVENTDATA and not otherwise allowed.
	scode: A-type address.
,VIEWFULL=*fcode*	Optional with EVENTDATA and not otherwise allowed.
	fcode: A-type address.
,COMPONENTDATA=*cdata*	Optional with EVENTDATA and not otherwise allowed.
	cdata: A-type address.
,EXCEPTION	Optional choice with EVENTDATA and not otherwise allowed.
,NOEXCEPTION	

Parameters

The parameters are explained as follows:

MAP
Specifies that a map of a format table is to be generated.

TABLEDATA=*tabname*
Specifies that the definition of an initialized format table is to be started. When you specify TABLEDATA, you also specify the name to be associated with the table and certain data that appears only once in the table.

EVENTDATA=*eventid*
Specifies the event identifier that is associated with a component trace event.

TABLEEND
Specifies the end of the definition of the format table.

,LOCBUFNAME=*bufname*
Specifies the name of the locate buffer exit routine that is loaded by the IPCS CTRACE subcommand. IPCS calls this routine to locate a component's trace buffers in a dump.

,LOCBUFADDR=*bufaddr*
Specifies the address of the locate buffer exit routine. IPCS calls this routine to locate a component's trace buffers in a dump.

,FILTERNAME=*pgmname*
Specifies the name of the component filter exit routine that is loaded by the IPCS CTRACE subcommand. IPCS calls this routine to provide component-specific filtering for that component's trace entries. No component filter exit is supplied if you do not specify one.

,FILTERADDR=*pgmaddr*
Specifies the address of the component filter exit routine. IPCS calls this routine to provide component-specific filtering for that component's trace entries. No component filter exit is supplied if you do not specify one.

,ENTRYLENGTH=*elength*
When *elength* is not zero, this parameter specifies the length of the fixed-length component trace entries that the component maintains. When *elength* is zero, it indicates that the component trace entries vary in length. A default of zero is assumed.

,MNEMONIC=*mnemonic*
Specifies a mnemonic name for the type of event being described. This name is the first information to be formatted on a line associated with an event entry of this type. The name permits the reader of formatted component traces to rapidly scan the output for patterns of events and events of particular interest.

,DESCRIPTION=*text*
Specifies descriptive, literal text to be associated with the type of trace entry being described. When this type of trace entry is formatted, the text appears at the end of the first line of the output. It helps the reader of the output to understand the significance of an entry, without having to access separate reference materials.

,MODELNAME=*modelname*
Specifies the name of the model that is to be used to format this trace entry. No model is used if MODELNAME or MODELADDR is not specified.

,MODELADDR=*modeladdr*
Specifies the address of the model to be used to format this trace entry. No model is used if MODELADDR or MODELNAME is not specified.

,FORMATNAME=*pgmname*
Specifies the name of the formatter routine that formats this trace entry. No formatter routine is called if FORMATNAME or FORMATADDR is not specified.

,FORMATADDR=*pgmaddr*
Specifies the address of the formatter routine that formats this trace entry. No formatter routine is called if FORMATADDR or FORMATNAME is not specified.

,OFFSETASID=*(ids)*
If you want ASID filtering to be performed (as requested by an IPCS CTRACE subcommand), use this parameter to specify the offsets to the ASID fields. The ASID fields occur at various offsets in the trace entry. Specify up to 5 offsets. An offset value may not exceed decimal 65,535. If you do not specify OFFSETASID, ASID filtering is not performed

,OFFSETJOBNAME=*(offsets)*
If you want job name filtering to be performed (as requested by an IPCS CTRACE subcommand), use this parameter to specify the offsets to the job name fields. The job name fields occur at various offsets in the trace entry. Specify up to 5 offsets. An offset value may not exceed decimal 65,535. If you do not specify OFFSETJOBNAME, job name filtering is not performed.

,VIEWSUMMARY=*scode*
Specifies the halfword view that the model processor uses to format summary fields from the trace entry. A default of X'8000' for *scode* is used if you do not specify this parameter.

,VIEWFULL=*fcode*
Specifies a halfword view (used by model processor) to format all fields from the trace entry. A default of X'0200' for *fcode* is used if you do not specify this parameter.

,COMPONENTDATA=*cdata*
This parameter is reserved for use by the component. If this parameter is not specified, a default of zero is assumed for *cdata* indicating that no component data is associated with the trace entry.

,EXCEPTION
,NOEXCEPTION

EXCEPTION specifies that this trace entry records an exceptional event. When the IPCS CTRACE subcommand is invoked with the EXCEPTION filtering option, only trace entries with the EXCEPTION attribute are formatted.

NOEXCEPTION specifies that the trace entries being described record normal events. These entries will not be formatted when the IPCS CTRACE subcommand is invoked with the EXCEPTION of the filtering option. The default is NOEXCEPTION.

Return and reason codes

None.

Chapter 141. ITTWRITE — Write a full trace buffer to DASD or tape

Description

The ITTWRITE macro enables the component trace external writer to write a full trace buffer out to a trace data set on DASD or tape.

The ITTWRITE macro asynchronously captures a full trace buffer while the application continues processing and writing trace entries to another trace buffer.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW key 0-7
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	64-bit.
ASC mode:	Primary or access register.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	No locks may be held.
Control Parameters:	Must be in the 64-bit primary address space.

Programming requirements

None.

Restrictions

If either the BUFFALET or the TBWCALET identifies the secondary or home address space, then both must identify the same address space (that is, both the trace buffer and the trace buffer writer control area must be in the same address space).

Register information

All registers are viewed as 64-bit values. After the caller issues the macro, the system might use some registers as work registers or might change the contents of some registers. When the system returns control to the caller, the contents of these registers are not the same as they were before the caller issued the macro. Therefore, if the caller depends on these registers containing the same value before and after issuing the macro, the caller must save these registers before issuing the macro and restore them after the system returns control.

When control returns to the caller, the general purpose registers (GPRs) contain:

Register
Contents

0
If GPR 15 contains 0 or 4, GPR 0 is used as a work register by the system; otherwise, GPR 0 contains a reason code.

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Performance implications

None.

Syntax

The standard form of the ITTWRITE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ITTWRITE.
ITTWRITE	
ƀ	One or more blanks must follow ITTWRITE.
BUFFADDR=*buffer_address*	*buffer_address*: RS-type address or register (2)-(12).
,BUFFALET=*buffer_alet*	*buffer_alet*: RS-type address or register (2)-(12).
,BUFFALET=NOBUFFALET	**Default:** BUFFALET=NOBUFFALET
,BUFFLEN=*buffer_length*	*buffer_length*: RS-type address or register (2)-(12).
,TOKEN=*token*	*token*: RS-type address or register (2)-(12).
,TBWCADDR=*tbwc_address*	*tbwc_address*: RS-type address or register (2)-(12).

Syntax	Description
,TBWCADDR64=*tbwc_address64*	*tbwc_address64*: RS-type address or register (2)-(12).
,TBWCALET=*tbwc_alet*	*tbwc_alet*: RS-type address or register (2)-(12).
,TBWCALET=NOTBWCALET	**Default:** TBWCALET=NOTBWCALET
,SYNCH=YES | NO	**Default:** SYNCH=NO
,RC=*return_code*	*return_code*: RS-type address or register (2)-(12).
,RSNCODE=*reason_code*	*reason_code*: RS-type address or register (2)-(12).
,COM=*comment*	*comment*: A comment string.
,COM=NULL	**Default:** COM=NULL.
,MF=(S)	**Default:** MF=(S)

Parameters

The parameters are explained as follows:

BUFFADDR=*buffer_address*
Specifies a required parameter that points to the address of the buffer to be written externally.

,BUFFALET=*buffer_alet*
,BUFFALET=NOBUFFALET
Contains the PASN ALET that identifies the address/data space where the buffer resides. Use this optional parameter when the buffer to be written externally resides in either a data space or an address space that is different from the current primary address space. The default is BUFFALET=NOBUFFALET.

,BUFFLEN=*buffer_length*
Specifies a required parameter that indicates the number of bytes in length of the buffer to be written externally. Though the buffer length is 64-bits, it is required to keep the buffer size within manageable limits. IBM suggest that the length be between 4KB and 512M. Component trace splits buffers that are too large to fit into a single block.

,TOKEN=*token*
Specifies a required parameter that specifies the token passed to the start/stop exit routine when it was requested to start tracing externally.

TBWCADDR=*tbwc_address*
Specifies a required parameter that points to a word that points to the address of the storage obtained by the application for the trace buffer writer control area (TBWC) mapped by ITTTBWC. The TBWC provides communication between the application and component trace. See TBWC in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete field names and lengths, offsets, and descriptions of the fields of the TBWC.

TBWCADDR64=*tbwc_address64*
Specifies a required parameter that points to a word that points to the address of the storage obtained by the application for the trace buffer writer control area (TBWC) mapped by ITTTBWC. The TBWC

provides communication between the application and component trace. See TBWC in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete field names and lengths, offsets, and descriptions of the fields of the TBWC.

,TBWCALET=*tbwc_alet*
,TBWCALET=NOTBWCALET
Contains the ALET that identifies the address/data space where the TBWC resides. Use this optional parameter when the TBWC resides in either a data space or an address space that is different from the current primary address space. The default is TBWCALET=NOTBWCALET.

,SYNCH=YES | NO
YES causes CTRACE to copy the application's buffers before control is returned instead of scheduling an asynchronous SRB to copy the buffer. The ITTWRITE function executes synchronously. The SYNCH keyword is optional. NO causes the ITTWRITE function to execute asynchronously.

Note: Because your application runs slower, IBM does not suggest that you use the SYNCH keyword on every ITTWRITE invocation. Use the SYNCH keyword in the start/stop routine any time that the trace buffers are to be freed. For example, when the trace is being turned off or the buffer size is changing, you can free trace buffer storage after issuing the ITTWRITE macro with the SYNCH keyword. The system copies the buffers to I/O buffers that CTRACE then can write to the external data set. The default is SYNCH=NO.

,RC=*return_code*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*reason_code*
Specifies the location where the system is to store the reason code. If GPR 15 contains a return code other than 0 or 4, the reason code is also in GPR 0.

,COM=*comment*
,COM=NULL
Comments the macro invocation. If the comment contains any lowercase characters, it must be enclosed in quotation marks.

,MF=(S)
Specifies the standard form of the ITTWRITE macro.

ABEND codes

The following table identifies abend code and reason code combinations, and a description of what each means:

Table 126. Abend codes for the ITTWRITE Macro

Abend Code	Reason Code	Description
00D	00010100	For the ITTWRITE macro, the parameter list version number is not correct.
00D	00010200	The system found either nonzero values in the reserved fields or unused fields for the requested service in the ITTWRITE macro parameter list.
00D	00010300	The buffer length passed was 0 or less.
00D	00010400	The buffer length is unusually large and is not supported by CTRACE.

Return and reason codes

When control returns from ITTWRITE, GPR 15 (and *return_code*, if you coded RC) contains one of the following return codes. The third byte of GPR 0 (and *reason_code*, if you coded RSNCODE) might contain one of the following reason codes.

Note: An application should always check the return code from the ITTWRITE macro. A non-zero code indicates that some data might have been lost in the next record output.

Table 127. Return and Reason Codes for the ITTWRITE Macro		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning**
00	None.	ITTWRITE was successful.
04	None.	ITTWRITE was unsuccessful. No data was captured because the trace is not connected to an active external writer.
08	xxxx01xx	Storage required to perform the write operation could not be obtained.
08	xxxx02xx	ITTWRITE was unable to schedule an SRB to process this request.
08	xxxx03xx	The control information (TBWC) has already been reused by the application.
0C	xxxx01xx	The caller is holding locks.
0C	xxxx02xx	The input token was not valid.
0C	xxxx0300	The TBWC is not valid because the sequence number is the same as a previous write request.
0C	xxxx0301	The TBWC is not valid for one of the following reasons: • The TBWC is not in central storage and the ITTWRITE issuer is disabled. • The BUFFALET is not the same as the TBWCALET.

Example

Indicate to component trace that the buffer at address TRACEADR is ready to be written out. Pass the token (TCWTRTKN) that the application received from the start/stop routine. Component trace is to store the return and reason codes from the ITTWRITE macro in TCRCODE and TCRSNCODE.

```
         ITTWRITE BUFFADDR=TRACEADR,BUFFLEN=TRACESIZ,                  X
               TOKEN=TCWTRTKN,TBWCADDR=TBWCADR,                        X
               RC=TCRCODE,RSNCODE=TCRSNCODE

         TBWCADR   DS A                TBWC address
         TRACEADR  DS A                Trace buffer address
         TRACESIZ  DS F                Trace buffer size
         TCWTRTKN  DS CL8              Trace writer token produced by
*                                      CTRACE upon connection
         TCRCODE   DS F                Return code from CTRACE
         TCRSNCODE DS F                Reason code from CTRACE
```

ITTWRITE - List form

Syntax

The list form of the ITTWRITE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ITTWRITE.
ITTWRITE	
ƀ	One or more blanks must follow ITTWRITE.

Syntax	Description
,MF=(L,*cntl)*	*cntl*: Symbol.
,MF=(L,*cntl,attr)*	*attr*: 1- to 60-character input string.
,MF=(L,*cntl*,0D)	**Default:** 0D

Parameters

The parameters are explained as follows:

,MF=(L,*cntl*)
,MF=(L,*cntl,attr*)
,MF=(L,*cntl*,0D)
Specifies the list form of the macro.

cntl is the name of a storage area for the parameter list.

attr is an optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

ITTWRITE - Execute form

Use the execute form of the ITTWRITE macro together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the ITTWRITE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ITTWRITE.
ITTWRITE	
ƀ	One or more blanks must follow ITTWRITE.
BUFFADDR=*buffer_address*	*buffer_address*: RS-type address or register (2)-(12).
,BUFFALET=*buffer_alet*	*buffer_alet*: RS-type address or register (2)-(12).
,BUFFALET=NOBUFFALET	**Default:** BUFFALET=NOBUFFALET

Syntax	Description
,BUFFLEN=*buffer_length*	*buffer_length*: RS-type address or register (2)-(12).
,TOKEN=*token*	*token*: RS-type address or register (2)-(12).
,TBWCADDR=*tbwc_address*	*tbwc_address*: RS-type address or register (2)-(12).
,TBWCADDR64=*tbwc_address64*	*tbwc_address64*: RS-type address or register (2)-(12).
,TBWCALET=*tbwc_alet*	*tbwc_alet*: RS-type address or register (2)-(12).
,TBWCALET=NOTBWCALET	**Default:** TBWCALET=NOTBWCALET
,SYNCH=YES | NO	**Default:** SYNCH=NO
,RC=*return_code*	*return_code*: RS-type address or register (2)-(12).
,RSNCODE=*reason_code*	*reason_code*: RS-type address or register (2)-(12).
,COM=*comment*	*comment*: A comment string.
,COM=NULL	**Default:** COM=NULL.
,MF=(E,*cntl*)	*cntl*: RX-type address or register (2) - (12).
,MF=(E,*cntl*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained under the standard form of the ITTWRITE macro with the following exception:

,MF=(E,*cntl*)
,MF=(E,*cntl*,COMPLETE)
Specifies the execute form of the macro.

cntl is the name of a storage area for the parameter list.

COMPLETE specifies that the system is to check the macro parameter syntax and supply defaults on parameters that you do not use. COMPLETE is the default.

Chapter 142. ITZXFILT — Transaction trace filter exit

Description

The ITZXFILT macro is used to invoke the Transaction Trace filter exit.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. PSW key 8 - 15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked or locked
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

1. Addressability to CVT and IHAECVT must be established prior to using this macro.
2. FRRs are allowed.
3. The version of the IWMCLSFY parameter list must be 4 or higher.

Input register information

Before issuing the ITZXFILT macro, the caller must insure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
The address of a 72-byte standard save area in the primary address space

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Unpredictable (Used as a work register by the system)

2-13
Unchanged

14
Unpredictable (Used as a work register by the system)

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Unpredictable (Used as a work register by the system)

2-13
Unchanged

14-15
Unpredictable (Used as a work register by the system)

Some callers depend on register contents remaining the same before and after issuing a macro. If the macro changes the contents of registers on which the caller depends, the caller must save them before issuing the macro and restore them after the macro returns control.

Performance implications

None.

Syntax

The ITZXFILT macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␢	One or more blanks must precede ITZXFILT.
ITZXFILT	
␢	One or more blanks must follow ITZXFILT.
FILTPARM=*filtparm*	*filtparm*: RS-type address or address in register (2) - (12).
,WKAREA=*wkarea*	*wkarea*: RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type address or address in register (2) - (12).

Parameters

The parameters are explained as follows:

name
This is an optional symbol, starting in column 1, that is the name on the ITZXFILT macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

FILTPARM=*filtparm*
This is a required input parameter of the IWMCLSFY parameter list to be passed to the transaction trace filter exit.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,WKAREA=*wkarea*
This is a required input parameter of a 256-byte work area to be used by the transaction trace filter exit routine.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,RETCODE=*retcode*
This is an optional output parameter into which the return code is to be copied from GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

ABEND codes

None.

Return and reason codes

When the ITZXFILT macro returns control to your program, GPR 15 (and *retcode,* when you code RETCODE) contains a return code.

The following table identifies the hexadecimal return and reason codes.

Table 128. Return and Reason Codes for the ITZXFILT Macro

Return Code	Meaning and Action
0	**Meaning**: The transaction trace token was created. **Action**: None.
4	**Meaning**: The transaction trace token was not created. **Action**: If the address of the 32-character transaction trace token field was not provided, provide it in the parameter list. Otherwise, the token was not created because of no match between the input parameter attributes and the transaction trace filter sets attributes.

Example

```
          IWMCLSFY MF=(L,TT_PARM),PLISTVER=MAX
          USING TT_PARM,R5
          LA    R1,TRANNAME
          ST    R1,TT_PARM_XTRXNAME_ADDR
          LA    R1,TRANTOKN
          ST    R1,TT_PARM_XTTRACETOKEN_ADDR
          ITZXFILT FILTPARM=TT_PARM,WKAREA=TT_WORK
          SPACE 1
          LTR   R15,R15
          BC    NZERO,NOTOKEN
          .
          .
NOTOKEN   DS    0H
          .
          .
TT_WORK   DS    CL256
TRANNAME  DC    CL8'MYWORKNM'
TRANTOKN  DC    CL32' '
```

Chapter 143. IXGBRWSE — Browse/read a log stream

Description

Use the IXGBRWSE macro to read and browse a log stream for log block information. Using IXGBRWSE, a program can read consecutive log blocks in a log stream or search for and read a specific log block in a log stream. IXGBRWSE returns the specified log block in the calling program's output buffer.

The requests for IXGBRWSE are:

- REQUEST=START, which starts a browse session. A browse session is identified by a browse token which is created by the browse start request. The browse session remains active until it is ended as a result of a REQUEST=END request or the log stream has been disconnected. See "REQUEST=START option of IXGBRWSE" on page 1354 for the syntax of this request.
- REQUEST=READCURSOR, which reads the next consecutive log block (or blocks) in the log stream. Use this request multiple times or use the MULTIBLOCK keyword to read consecutive blocks in a log stream. See "REQUEST=READCURSOR option of IXGBRWSE" on page 1360 for the syntax of this request.
- REQUEST=READBLOCK, which reads a selected log block in a log stream. See "REQUEST=READBLOCK option of IXGBRWSE" on page 1366 for the syntax of this request.
- REQUEST=RESET, which resets the browse cursor to either the beginning or the end of the log stream. See "REQUEST=RESET option of IXGBRWSE" on page 1372 for the syntax of this request.
- REQUEST=END, which ends a browse session. See "REQUEST=END option of IXGBRWSE" on page 1377 for the syntax of this request.

For information about using the system logger services and the IXGBRWSE request, see *z/OS MVS Programming: Assembler Services Guide*, which also includes information about related macros IXGCONN, IXGINVNT, IXGWRITE, IXGDELET, and IXGQUERY.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem or Supervisor state with any PSW key. The caller must be in supervisor state with any system (0-7) PSW key to either invoke this service in SRB mode or to use the MODE=SYNCEXIT keyword.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, HASN or SASN
AMODE:	31-bit or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.

Environmental factor	Requirement
Control parameters:	All control parameters must be in the primary address space with the following exceptions: • The ECB should be addressable from the home address space. • Any parameter area that is explicitly ALET-qualified as allowed by the input parameter (for example, the area referenced by the BUFFER parameter when the BUFFALET parameter is specified) must be in an address or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All storage areas specified must be in the same storage key as the caller with the following exception: • Any parameter area is explicitly storage key qualified as allowed by the input parameters (example: the area referenced by the BUFFER parameter when the BUFFKEY parameter is also specified).

Programming requirements

- The current primary address space must be the same primary address space used at the time your program issued the IXGCONN request.
- The calling program must be connected to the log stream through the IXGCONN service with either read or write authority.
- The parameter list for this service must be addressable in the caller's primary address space.
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include macro IXGANSAA in your program. This macro maps the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- For a READCURSOR browse request with the MULTIBLOCK=YES option, include the IXGBRMLT mapping macro in your program. This macro provides a mapping of the area returned by the system logger for each block that is returned in the caller's buffer. Additionally, the area pointed to by the BUFFER or BUFFER64 parameter must be on a word boundary for multiple log block READCURSOR requests.
- Although the data pointed to by the BUFFER64 keyword may be above the bar (2-gigabyte), the length of the name or address of the input field specified in the BUFFLEN keyword is still limited to 4 bytes.
- When coding the MODE=SYNCECB and ECB parameters, you must ensure that:
  - The virtual storage area specified for the ECB resides on a fullword boundary.
  - You initialize the ECB field to zero.
  - The ECB resides in either common or home address space storage at the time the IXGBRWSE request is issued.
  - The storage used for output parameters, such as ANSAREA, BROWSETOKEN, BUFFER, BUFFER64, ANSAREA, BLKSIZE, TIMESTAMP, and RETBLOCKID, are accessible by both the IXGBRWSE invoker and the ECB waiter.
- When coding the MODE=SYNCEXIT parameter, you must ensure that the storage used for output parameters, such as ANSAREA, BROWSETOKEN, BUFFER, BUFFER64, ANSAREA, BLKSIZE, TIMESTAMP, and RETBLOCKID, are accessible by both the IXGBRWSE invoker and the completion exit routine.

Restrictions

There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.

You can call any of the system logger services in either AMODE 31 or 64, but the parameter list and all other data addresses, with the excption of BUFFER64 must reside in 31-bit storage.

Input register information

Before issuing the IXGBRWSE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if register 15 contains a non-zero return code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

When control returns to a caller running in AMODE 64, the 64–bit registers contain:

Register
Contents

0-1
Used as a work register by the system, if the caller specified BUFFER64. Otherwise, unchanged.

2-13
Unchanged

14
Unchanged

15
Used as a work register by the system.

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

REQUEST=START option of IXGBRWSE

The IXGBRWSE macro with the REQUEST=START parameter starts a browse session and sets the starting position of the browse cursor.

Syntax for REQUEST=START

The IXGBRWSE REQUEST=START macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
ƀ	One or more blanks must follow IXGBRWSE.
REQUEST=START	
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BROWSETOKEN=*browsetoken*	*browsetoken*: RS-type address or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,OLDEST	**Default**: OLDEST
,YOUNGEST	
,STARTBLOCKID=*startblockid*	*startblockid*: RS-type address or register (2) - (12).
,SEARCH=*search*	*search*: RS-type address or register (2) - (12).
GMT=YES	

Syntax	Description
GMT=NO	
VIEW=ACTIVE	**Default**: VIEW=ACTIVE
VIEW=ALL	
VIEW=NO_VIEW	
MODE=SYNC	**Default**: MODE=SYNC
MODE=SYNCECB	
MODE=SYNCEXIT	
,REQDATA=*reqdata*	*reqdata*: RS-type address or register (2) - (12).
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,DIAG=NO_DIAG	**Default**: DIAG=NO_DIAG
,DIAG=NO	
,DIAG=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	

Syntax	Description
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters for REQUEST=START

The parameters are explained as follows:

REQUEST=START
Requests that a browse session be started.

,STREAMTOKEN=*streamtoken*
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to browse and read. The stream token is returned by the IXGCONN service at connection to the log stream.

,BROWSETOKEN=*browsetoken*
Specifies the name or address (using a register) of a required 4-byte output area where a token uniquely identifying the browse session is returned by the IXGBRWSE REQUEST=START request. This browse token is then used as an input to subsequent IXGBRWSE requests to identify the browse session.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,OLDEST
,YOUNGEST
,STARTBLOCKID=*startblockid*
,SEARCH=*search*
Specifies where the cursor should be set for the start of the browse session.

- OLDEST: Specifies that the block cursor be positioned at the oldest log block in the log stream.

  When VIEW=ACTIVE is specified for this browse session, the cursor is positioned at the oldest active log block in the log stream. If there is no active data in the log stream, the request will fail.

  When VIEW=ALL is specified, the cursor is positioned at the oldest log block in the log stream of the active and inactive data. If there is neither active nor inactive data in the log stream, the request will fail.
- YOUNGEST: Specifies that the block cursor be positioned at the youngest log block in the log stream.

  When VIEW=ACTIVE is specified for this browse session, the cursor is positioned at the youngest active log block in the log stream.

  When VIEW=ALL is specified, the cursor is positioned at the youngest log block in the log stream, even if the youngest block is eligible for deletion.
- STARTBLOCKID=*startblockid*: Specifies the name (or register) of a 8-byte input field containing the block identifier for the log block you want to use as the starting cursor position.

When VIEW=ALL is specified, you must specify a starting block that is active.

- SEARCH=*search*: Specifies the name (or register) of a 64-bit input field containing the time stamp you want to use in searching for a particular log block as the starting cursor position for this browse session. For information on how the SEARCH keyword works, see *z/OS MVS Programming: Assembler Services Guide*.

  The time stamp must be Coordinated universal time (UTC) or local time, in time of day (TOD) clock format. The GMT parameter is required with the SEARCH parameter.

,GMT=YES
,GMT=NO

Specifies whether the time stamp specified on the SEARCH parameter is UTC or local time.

- GMT=YES: The time stamp specified on the SEARCH parameter is in UTC format.
- GMT=NO: The time stamp specified on the SEARCH parameter is local time.

VIEW=ACTIVE
VIEW=ALL
VIEW=NO_VIEW

Specifies whether requests issued during this browse session return active data only, or both active and inactive data. Active data is data that has not been marked for deletion via the IXGDELET service. Inactive data is data that has been deleted via IXGDELET but has not been physically deleted from the log stream because of the retention period specified in the log stream policy definition in the active system logger couple data set.

- VIEW=ACTIVE, which is the default, specifies that in this browse session, system logger will only return active data from the log stream.
- VIEW=ALL specifies that in this browse session, system logger will return both active and inactive data.

  When VIEW=ALL is specified and a log block is returned, system logger sets a flag in the answer area, AnsaaBlkFromInactive, indicating whether the block was active or eligible for deletion.
- VIEW=NO_VIEW specifies that the default VIEW value will be used for the browse session.

,MODE=SYNC
,MODE=SYNCECB
,MODE=SYNCEXIT

Specifies that the request should be processed in one of the following ways:

- MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. If the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

  To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

,ECB=*ecb*
Specifies the name or address (using a register) of a 4-byte input field containing an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

,DIAG=NO_DIAG
,DIAG=NO
,DIAG=YES
Specifies whether or not the DIAG option on the IXGCONN for this logstream will be in effect for this browse session. Refer to the DIAG keyword on the IXGINVNT, IXGCONN, and IXGDELET macro services.

If you specify DIAG=NO_DIAG, which is the default, then the DIAG option on the IXGCONN for this logstream will be in effect for this browse session.

If you specify DIAG=NO, thenLogger will not take additional diagnostic action as defined in the logstream definition DIAG parameter.

If you specify DIAG=YES, then Logger will take additional diagnostic action as defined on the logstream definition DIAG parameter providing the IXGCONN connect DIAG specification allows it.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX

- A decimal value of 0, 1 or 2

,RETCODE=*retcode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

You should use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=READCURSOR option of IXGBRWSE

The IXGBRWSE macro with the REQUEST=READCURSOR option allows a program to read the next consecutive log block in a log stream. Subsequent READCURSOR requests will start reading at the next consecutive block. Use this request multiple times or use the MULTIBLOCK keyword to read a series of consecutive log blocks. The direction of the browse is controlled by the program and can be changed dynamically.

READCURSOR requests are limited to reading log blocks within the range of data defined by the browse session's view. The view is controlled by the VIEW keyword on either the browse START request or the browse RESET request.

Note: REQUEST=READCURSOR reads the next consecutive log block in the log stream, but the blocks may not be in exact local time sequence. This can happen, for example, because of daylight savings time, one or more records with the same local time stamp, or multiple applications writing to the same log stream.

Syntax for REQUEST=READCURSOR

The IXGBRWSE REQUEST=READCURSOR macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
ƀ	One or more blanks must follow IXGBRWSE.
REQUEST=READCURSOR	
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BROWSETOKEN=*browsetoken*	*browsetoken*: RS-type address or register (2) - (12).
,BUFFER=*buffer*	*buffer*: RS-type address or register (2) - (12).
,BUFFER64=*buffer64*	*buffer64*: RS-type address or register (2) - (12).
,BUFFLEN=*bufflen*	*bufflen*: RS-type address or register (2) - (12).
,DIRECTION=OLDTOYOUNG	
,DIRECTION=YOUNGTOOLD	
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).

Syntax	Description
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,BUFFKEY=*buffkey*	*buffkey*: RS-type address or register (2) - (12).
	Default: PSW key of the caller
,BUFFALET=*buffalet*	*buffalet*: RS-type address or register (2) - (12).
	Default: BUFFALET=0
,BLKSIZE=*blksize*	*blksize*: RS-type address or register (2) - (12). **Default**: BLKSIZE=0
,MULTIBLOCK=YES	
,MULTIBLOCK=NO	**Default**: MULTIBLOCK=NO
,RETBLOCKID=*retblockid*	*retblockid*: RS-type address or register (2) - (12). **Default**: NO_BLKID **Note**: RETBLOCKID is valid with MULTIBLOCK=NO only.
,TIMESTAMP=*timestamp*	*timestamp*: RS-type address or register (2) - (12). **Default**: NO_TIMESTAMP **Note**: TIMESTAMP is valid with MULTIBLOCK=NO only.
,RETBLOCKINFO=YES	
,RETBLOCKINFO=NO	**Default**: NO **Note**: RETBLOCKINFO is valid with MULTIBLOCK=YES only.
,MAXNUMLOGBLOCKS=*maxnumlogblocks*	
	maxnumlogblocks: RS-type address or register (2) - (12).
	Default: MAXNUMLOGBLOCKS=0 **Note**: MAXNUMLOGBLOCKS is valid with MULTIBLOCK=YES only.
MODE=SYNC	**Default**: MODE=SYNC
MODE=SYNCECB	
MODE=SYNCEXIT	
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).

Syntax	Description
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters for REQUEST=READCURSOR

The parameters are explained as follows:

REQUEST=READCURSOR
Requests that a program read the next consecutive log block in the log stream, in the direction specified on the DIRECTION parameter.

,STREAMTOKEN=*streamtoken*
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to browse and read. The stream token is returned by the IXGCONN service at connection to the log stream.

,BROWSETOKEN=*browsetoken*
Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned on the IXGBRWSE REQUEST=START request.

,BUFFER=*buffer*
,BUFFER64=*buffer64*
Specifies the name or address (using a register) of a required output field that contains the buffer into which the log block is read.

- BUFFER=*buffer* specifies that the location of the buffer is in 31-bit storage.
- BUFFER64=*buffer64* specifies that the location of the buffer is in 64-bit storage.

the BUFFER and BUFFER64 parameters are mutually exclusive.

,BUFFLEN=*bufflen*
Specifies the name or address (using a register) of a required 4-byte input field that contains the length of the buffer specified on the BUFFER or BUFFER64 parameter.

IXGBRWSE will return the length of the block in the BLKSIZE parameter, if specified. If you specify MULTIBLOCK=NO, you can issue IXGBRWSE with BLKSIZE specified to obtain the length of the block and then re-issue IXGBRWSE using the returned BLKSIZE value in the BUFFLEN parameter.

,DIRECTION=OLDTOYOUNG
,DIRECTION=YOUNGTOOLD
Specifies the direction that you want the cursor to move to read the next consecutive log block. Specify OLDTOYOUNG to get the next youngest block or YOUNGTOOLD to get the next oldest block.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,BUFFKEY=*buffkey*
Specifies the name (or address in a register) of a 4-byte input field specifying the storage key for the buffer specified on the BUFFER or BUFFER64 parameter.

If the caller is running in problem state, the caller's PSW key and the key specified in the BUFFKEY parameter must match.

If the caller is running in supervisor state, specify any syntactically valid (0 through 15) key on the BUFFKEY parameter.

If you omit the BUFFKEY parameter, the default used is the PSW key of the caller.

,BUFFALET=*buffalet*
Specifies the name (or address in a register) of a 4-byte input field specifying the access list entry table (ALET) to be used to access the buffer specified on the BUFFER or BUFFER64 keyword. If the buffer is ALET-qualified, the ALET must index a valid entry on the task's dispatchable unit access list (DUAL) or specify a SCOPE=COMMON data space. An ALET that indexes the system logger PASN-AL list will not work.

The default is 0, which means that the buffer is in the calling program's primary address space.

,BLKSIZE=*blksize*
Specifies the name or address (using a register) of a 4-byte output field where the space used or needed in the BUFFER or BUFFER64 area is returned. When MULTIBLOCK=NO is specified and there is enough space in the buffer to return the requested log block data, the actual size of the log block is returned. When MULTIBLOCK=YES is specified and there is enough space in the buffer to return the requested log blocks, the amount of space used in the BUFFER or BUFFER64 area is returned. If the BUFFLEN value is not large enough to allow any log block data to be returned, then the BLKSIZE value will indicate the minimum amount of space necessary to return the next log block.

,MULTIBLOCK=YES
,MULTIBLOCK=NO
Specifies whether one or more than one log stream log block will be returned by the read cursor request.

- MULTIBLOCK=NO indicates that only one log stream log block is to be returned.
- MULTIBLOCK=YES indicates that the system logger will retrieve as many log blocks as meet the browse parameter criteria and fit into the caller's buffer.

,RETBLOCKID=*retblockid*
Specifies the name or address (using a register) of an 8-byte output field where the identifier or the requested log block is returned

,TIMESTAMP=*timestamp*
Specifies the name or address (using a register) of a 16-byte output field where the Coordinated universal time stamp and the local time stamp associated with the requested log block are returned. The UTC time stamp is first, then the local time stamp. Both time stamps are in TOD-clock format.

,RETBLOCKINFO=YES
,RETBLOCKINFO=NO
Specifies whether or not system logger should return the log blocksize, blockid, timestamps and other identification information in the caller's buffer as part of the output. Specify RETBLOCKINFO=YES to receive each log block's identification information. Specify RETBLOCKINFO=NO to only receive the information necessary to navigate the caller's buffer.

If you omit the RETBLOCKINFO parameter, RETBLOCKINFO=NO is the default.

,MAXNUMLOGBLOCKS=*xmaxnumlogblocks*
Specifies the name (or address in a register) of an optional fullword input that indicates the maximum number of log blocks to be returned in the buffer. When a non-zero value is specified, system logger will not return more than this requested number of log blocks, even if there are more log blocks that meet the other browse parameter criteria.

- If enough room is provided in the BUFFLEN value and there are sufficient log blocks that meet the browse criteria, system logger will return the requested maximum number of log blocks.
- If enough room is not provided in the BUFFLEN value, system logger will return as many log blocks as fit into the caller's buffer.
- If there are fewer log blocks remaining than the requested maximum number, system logger will return as many of the remaining log blocks as fit into the caller's buffer.

If you omit the MAXNUMLOGBLOCKS, the default is 0.

,MODE=SYNC
,MODE=SYNCECB
,MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:

- MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. if the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

  To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

ECB=*ecb*
Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2

,RETCODE=*retcode*

Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*

Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=READBLOCK option of IXGBRWSE

The IXGBRWSE macro with the REQUEST=READBLOCK parameter allows a program to search for and read a specific log block from the log stream. The target can be defined either by the log block identifier or by a time stamp.

Syntax for REQUEST=READBLOCK

The IXGBRWSE REQUEST=READBLOCK macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
ƀ	One or more blanks must follow IXGBRWSE.
REQUEST=READBLOCK	
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BROWSETOKEN=*browsetoken*	*browsetoken*: RS-type address or register (2) - (12).
,BLOCKID=*blockid*	*blockid*: RS-type address or register (2) - (12).
,SEARCH=*search*	*search*: RS-type address or register (2) - (12).
,BUFFER=*buffer*	*buffer*: RS-type address or register (2) - (12).
,BUFFER64=*buffer64*	*buffer64*: RS-type address or register (2) - (12).
,BUFFLEN=*bufflen*	*bufflen*: RS-type address or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
GMT=YES	
GMT=NO	
,BUFFKEY=*buffkey*	*buffkey*: RS-type address or register (2) - (12).
	Default: PSW key of the caller
,BUFFALET=*buffalet*	*buffalet*: RS-type address or register (2) - (12).
	Default: BUFFALET=0

Syntax	Description
,BLKSIZE=*blksize*	*blksize*: RS-type address or register (2) - (12).
	Default: BLKSIZE=0
,RETBLOCKID=*retblockid*	*retblockid*: RS-type address or register (2) - (12).
	Default: NO_BLKID
,TIMESTAMP=*timestamp*	*timestamp*: RS-type address or register (2) - (12).
	Default: NO_TIMESTAMP
MODE=SYNC	**Default**: MODE=SYNC
MODE=SYNCECB	
MODE=SYNCEXIT	
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters for REQUEST=READBLOCK

The parameters are explained as follows:

REQUEST=READBLOCK
Requests that a program read a specific block from the log stream. The target can be defined either by the log block identifier or by a time stamp.

,STREAMTOKEN=*streamtoken*
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

,BROWSETOKEN=*browsetoken*
Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned from the IXGBRWSE REQUEST=START request.

,BLOCKID=*blockid*
Specifies the name or address (using a register) of an 8-byte input field that contains the block identifier of the log block you wish to read. The block identifier was returned from the IXGWRITE request.

,SEARCH=*search*
Specifies the name or address (using a register) of a 64-bit input field containing the time stamp for the log block you wish to search for and read. The time stamp must be Greenwich mean time or local time,

When you use a time stamp as a search criteria, IXGBRWSE searches in the oldest-to-youngest direction, searching for a log block with an exactly matching time stamp. If no exact match is found, IXGBRWSE reads the next latest (youngest) time stamp. For information on how the SEARCH keyword works, see *z/OS MVS Programming: Assembler Services Guide*.

The GMT parameter is required with the SEARCH parameter.

,BUFFER=*buffer*
,BUFFER64=*buffer64*
Specifies the name or address (using a register) of a required output field that contains the buffer into which the log block is read.

- BUFFER=*buffer* specifies that the location of the buffer is in 31-bit storage.
- BUFFER64=*buffer64* specifies that the location of the buffer is in 64-bit storage.

the BUFFER and BUFFER64 parameters are mutually exclusive.

,BUFFLEN=*bufflen*
Specifies the name or address (using a register) of a required 4-byte input field that contains the length of the buffer specified on the BUFFER or BUFFER64 parameter.

IXGBRWSE will return the length of the block in the BLKSIZE parameter, if specified. You can issue IXGBRWSE with BLKSIZE specified to obtain the length of the block and then re-issue IXGBRWSE using the returned BLKSIZE value in the BUFFLEN parameter.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 32 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area size, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,GMT=YES
,GMT=NO
Specifies whether the time stamp specified on the SEARCH parameter is in Coordinated universal time (UTC) or local time.

- GMT=YES: The time stamp specified on the SEARCH parameter is in Greenwich mean time.
- GMT=NO: The time stamp specified on the SEARCH parameter is local time.

,BUFFKEY=*buffkey*
Specifies the name (or address in a register) of a 4-byte input field specifying the storage key for the buffer specified on the BUFFER or BUFFER64 parameter.

If the caller is running in problem state, the caller's PSW key and the key specified in the BUFFKEY parameter must match.

If the caller is running in supervisor state, specify any syntactically valid (0 through 15) key on the BUFFKEY parameter.

If you omit the BUFFKEY parameter, the default used is the PSW key of the caller.

,BUFFALET=*buffalet*
Specifies the name (or address in a register) of a 4-byte input field specifying the access list entry table (ALET) to be used to access the buffer specified on the BUFFER or BUFFER64 keyword. If the buffer is ALET-qualified, the ALET must index a valid entry on the task's dispatchable unit access list (DUAL) or specify a SCOPE=COMMON data space. An ALET that indexes the system logger PASN-AL list will not work.

The default is 0, which means that the buffer is in the calling program's primary address space.

,BLKSIZE=*blksize*
Specifies the name or address (using a register) of a 4-byte output field where the actual size of the requested log block is returned.

,RETBLOCKID=*retblockid*
Specifies the name or address (using a register) of a 8-byte output field where the identifier of the requested log block is returned.

,TIMESTAMP=*timestamp*
Specifies the name or address (using a register) of a 16-byte output field where the Coordinated universal time and local time stamps associated with the requested log block is returned. The UTC time stamp is first, then the local time stamp. Both time stamps will be in TOD-clock format.

,MODE=SYNC
,MODE=SYNCECB
,MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:

- MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. if the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter

on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

ECB=*ecb*
Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2

,RETCODE=*retcode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=RESET option of IXGBRWSE

The IXGBRWSE macro with the REQUEST=RESET parameter allows a program to re-position the browse cursor to either the youngest or oldest block in the log stream.

Syntax for REQUEST=RESET

The IXGBRWSE REQUEST=RESET macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
ƀ	One or more blanks must follow IXGBRWSE.
REQUEST=RESET	
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BROWSETOKEN=*browsetoken*	*browsetoken*: RS-type address or register (2) - (12).
,POSITION=YOUNGEST	
,POSITION=OLDEST	
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
VIEW=ACTIVE	
VIEW=ALL	
MODE=SYNC	**Default**: MODE=SYNC
MODE=SYNCECB	
MODE=SYNCEXIT	
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	

Syntax	Description
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters for REQUEST=RESET

The parameters are explained as follows:

REQUEST=RESET
Requests that the browse cursor be repositioned at either the oldest or youngest block in the log stream.

,STREAMTOKEN=*streamtoken*
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

,BROWSETOKEN=*browsetoken*
Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned from the IXGBRWSE REQUEST=START request.

,POSITION=YOUNGEST
,POSITION=OLDEST
Specifies the cursor position desired, at either the youngest or the oldest log block in the log stream.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 32 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area size, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

VIEW=ACTIVE
VIEW=ALL
Specifies whether requests issued during this browse session return active data only, or both active and inactive data. Active data is data that has not been marked for deletion via the IXGDELET service. Inactive data is data that has been deleted via IXGDELET but has not been physically deleted from the log stream because of the retention period specified in the log stream policy definition in the active system logger couple data set.

- VIEW=ACTIVE, which is the default, specifies that in this browse session, system logger will only return active data from the log stream.
- VIEW=ALL specifies that in this browse session, system logger will return both active and inactive data.

  When VIEW=ALL is specified and a log block is returned, system logger sets a flag in the answer area, AnsaaBlkFromInactive, indicating whether the block was active or eligible for deletion.

,MODE=SYNC
,MODE=SYNCECB
,MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:

- MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. if the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

  To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

ECB=*ecb*
Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2

,RETCODE=*retcode*

Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*

Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the

execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

REQUEST=END option of IXGBRWSE

The IXGBRWSE macro with the REQUEST=END parameter ends the browse session begun with the REQUEST=START parameter.

Syntax for REQUEST=END

The IXGBRWSE REQUEST=END macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
ƀ	One or more blanks must follow IXGBRWSE.
REQUEST=END	

Syntax	Description
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BROWSETOKEN=*browsetoken*	*browsetoken*: RS-type address or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
MODE=SYNC	**Default**: MODE=SYNC
MODE=SYNCECB	
MODE=SYNCEXIT	
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters for REQUEST=END

The parameters are explained as follows:

REQUEST=END
Requests that the browse session be ended.

,STREAMTOKEN=*streamtoken*
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

,BROWSETOKEN=*browsetoken*
Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned from the IXGBRWSE REQUEST=START request.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,MODE=SYNC
,MODE=SYNCECB
,MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:

- MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.

  When a MODE=SYNCECB request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to BUFFER, BUFFER64, RETBLOCKID, TIMESTAMP, and BLKSIZE.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. if the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to BUFFER, BUFFER64, RETBLOCKID, TIMESTAMP, and BLKSIZE.

  To use this parameter, the system where the application is running must be IPLed. The application must run in supervisor state, key 0-7 to use this parameter.

ECB=*ecb*
Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.

- The ECB must reside in either common storage or the home address space at the time the IXGBRWSE request is issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports both the following parameters and parameters from version 0:
  - DIAG
  - REQDATA
- **2**, supports both the following parameters and parameters from version 0 and 1:
  - MAXNUMLOGBLOCKS
  - MULTIBLOCK
  - RETBLOCKINFO

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1 or 2

,RETCODE=*retcode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes

The IXGBRWSE service may issue abend X'1C5' with reason codes X'804', X'85F' or X'30006'. See *z/OS MVS System Codes* for more information on this abend.

Return and reason codes

When IXGBRWSE macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

Note: A program invoking the IXGBRWSE service may indicate via the MODE parameter that requests which can not be completed synchronously should have control returned to the caller prior to completion

of the request. When the request does complete, the invoker will be notified and the return and reason codes are in the answer area mapped by IXGANSAA.

The IXGCON mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

00
IXGRSNCODEOK - Service completes successfully.

04
IXGRSNCODEWARNING - Service completes with a warning.

08
IXGRETCODEERROR - Service does not complete.

0C
IXGRETCODECOMPERROR - Service does not complete.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 129. Return and Reason Codes for the IXGBRWSE Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**Equate Symbol**: IxgRsnCodeOk **Explanation**: Request processed successfully.
04	xxxx0401	**Equate Symbol**: IxgRsnCodeProcessedAsynch **Explanation**: Program error. The program specified MODE=SYNCECB and the request must be processed asynchronously. **Action**: Wait for the ECB specified on the ECB parameter to be posted, indicating that the request is complete. Check the ANSAA_ASYNCH_RETCODE and ANSAA_ASYNCH_RSNCODE fields, mapped by IXGANSAA, to determine whether the request completed successfully.
04	xxxx0402	**Equate Symbol**: IxgRsnCodeWarningDel **Explanation**: Environment error. The request completed successfully, but the data requested was deleted from the log stream. The next available data in the log stream in the direction specified is returned. **Action**: Determine whether this is an acceptable condition for your application. If so, ignore this condition. If not, provide serialization or some other installation protocol to prevent deletes from being performed by other applications on the log stream during a browse session.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0403	**Equate Symbol**: IxgRsnCodeWarningGap **Explanation**: Environment error. The request completed successfully, but the data requested was unreadable. The next readable data in the log stream in the specified direction is returned. This condition could be caused by either an I/O error while attempting to read a log data set or a log data set deleted without using the logger interfaces. **Action**: The action necessary is completely up to the application, depending on how critical your data is. You can do one of the following: • Accept this condition and continue reading. • Stop processing the log all together. • Attempt to get the problem rectified, if possible, and then attempt to re-read the log data.
04	xxxx0405	**Equate Symbol**: IxgRsnCodeWarningLossOfData **Explanation**: Environment error. Returned for READCURSOR, START OLDEST and RESET OLDEST requests. This condition occurs when a system and coupling facility fail and not all of the log data in the log stream could be recovered. • For READCURSOR: A log block has been returned, but there may be log blocks permanently missing between this log block and the one previously returned. • For START OLDEST and RESET OLDEST: The oldest log blocks in the log stream may be permanently missing, the browse cursor is set at the oldest available log block. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0416	**Equate Symbol**: IxgRsnCodeWarningMultiblock **Explanation**: Environment error. Returned for READCURSOR requests with MULTIBLOCK=YES specified only. The request completed successfully, which means that some log block data was returned, but at least one of the log blocks returned in the buffer area encountered a warning return code condition. To determine which log block or blocks encountered the warning condition, check the fields, Ixgbrmlt_RetCode and Ixgbrmlt_RsnCode, as the log blocks are processed by your program. **Action**: The action necessary is completely up to the application, depending on how critical your data is. You can do one of the following: • Accept this condition and continue reading. • Stop processing the log all together. • Attempt to get the problem rectified, if possible, and then attempt to re-read the log data.
04	xxxx0417	**Equate Symbol**: IxgRsnCodeMultiblockErrorWarning **Explanation**: Environment error. Returned for READCURSOR requests with MULTIBLOCK=YES specified only. A log block has been returned, but an error condition was encountered while attempting to read more data. This may be issued when some log block data is returned and an end of the log stream (eof) is reached. **Action**: The action necessary is completely up to the application, depending on how critical your data is. You can do one of the following: • Accept this condition and continue reading. • Stop processing the log all together. • Attempt to get the problem rectified, if possible, and then attempt to re-read the log data.
08	xxxx0801	**Equate Symbol**: IxgRsnCodeBadParmlist **Explanation**: Program error. The parameter list could not be accessed. **Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx0802	**Equate Symbol**: IxgRsnCodeXESError **Explanation**: System error. A severe cross-system extended services (XES) error has occurred. **Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0803	**Equate Symbol**: IxgRsnCodeBadBuffer **Explanation**: Program error. The virtual storage area specified on the BUFFER or BUFFER64 parameter is not addressable. On IXGBRWSE READCURSOR MULTIBLOCK requests, the buffer address must be on a word boundary. **Action**: Ensure that the storage area specified on the BUFFER or BUFFER64 parameter is accessible to system logger for the duration of the request. If the BUFFKEY parameter is specified, make sure it contains a valid key associated with the storage area. If BUFFKEY is not used, ensure that the storage is in the same key as the program at the time the logger service was requested. The storage must be addressable in the caller's primary address space. For IXGBRWSE READCURSOR MULTIBLOCK requests, put the buffer address on a word boundary.
08	xxxx0804	**Equate Symbol**: IxgRsnCodeNoBlock **Explanation**: Program error. The block identifier or time stamp does not exist in the requested view of the log stream. If the SEARCH parameter was specified on a START request, the time stamp is greater than any block in the log stream. Either the value provided was never a valid location within the log stream, or a prior IXGDELET request deleted the portion of the log stream it referred to. **Action**: Ensure that the value provided references an existing portion of the log stream.
08	xxxx0806	**Equate Symbol**: IxgRsnCodeBadStmToken **Explanation**: Program error. One of the following occurred: • The stream token was not valid. • The specified request was issued from an address space other than the connector's address space. **Action**: Do one of the following: • Make sure that the stream token specified is valid. • Ensure that the request was issued from the connector's address space.
08	xxxx0807	**Equate Symbol**: IxgRsnCodeBadBrwToken **Explanation**: Program error. The browse token specified is not valid. **Action**: Ensure that the browse token being passed to the IXGBRWSE service is the same one returned from the IXGBRWSE REQUEST=START function.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx080A	**Equate Symbol**: IxgRsnCodeRequestLocked **Explanation**: Program error. The program issuing the request is holding a lock. **Action**: Ensure that the program issuing the request is not holding a lock.
08	xxxx080F	**Equate Symbol**: IxgRsnCodeBadBufsize **Explanation**: Program error. The buffer specified on the BUFFER or BUFFER64 parameter is not large enough to contain the next log block. No data is returned. **Action**: Obtain a buffer of at least the length returned in the BLKSIZE parameter and then re-issue the request.
08	xxxx0814	**Equate Symbol**: IxgRsnCodeNotAvailForIPL **Explanation**: Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization. **Action**: See the explanation for system messages issued during system logger initialization.
08	xxxx0815	**Equate Symbol**: IxgRsnCodeNotEnabled **Explanation**: Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails. **Action**: Make sure the program issuing the request is enabled for I/O and external interrupts.
08	xxxx0816	**Equate Symbol**: IxgRsnCodeBadAnslen **Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro. **Action**: Re-issue the request, specifying an answer area of the required size.
08	xxxx0817	**Equate Symbol**: IxgRsnCodeBadAnsarea **Explanation**: Program error. The storage area specified on the ANSAREA parameter cannot be accessed. This may occur after the system logger address space has terminated. **Action**: Specify storage that is in the caller's primary address space and in the same key as the calling program at the time the system logger service was issued. This storage must be accessible until the request completes.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0818	**Equate Symbol**: IxgRsnCodeBadBlockidStor **Explanation**: Program error. The storage area specified by BLOCKID cannot be accessed. **Action**: Ensure that the storage area is accessible to system logger for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx081C	**Equate Symbol**: IxgRsnCodeNotAuthFunc **Explanation**: Program error. The browse function (IXGBRWSE) is not authorized for the log stream connection token. The program connected to the log stream with the AUTH=WRITE parameter and was granted limited access (for example, write only log data and disconnect) as per the defined security access profiles, and then tried to perform a log stream access service other than IXGWRITE or IXGCONN (to disconnect). **Action**: A connection to the log stream with the appropriate access authority must be established before using the log stream browse (IXGBRWSE) service. If AUTH=WRITE was already in use when the error was encountered, then check with your installation's security administrator on obtaining the appropriate access to the log stream.
08	xxxx082D	**Equate Symbol**: IxgRsnCodeExpiredStmToken **Explanation**: Environment error. The stream token is no longer valid because the connector has been disconnected. **Action**: Connect to the log stream again before issuing any functional requests.
08	xxxx0836	**Equate Symbol**: IxgRsnCodeBadGap **Explanation**: Environment error. The request failed because the requested log data was unreadable. This condition could be caused by either an I/O error while attempting to read a log data set or a log data set deleted without using logger interfaces. **Action**: For an IXGBRWSE request, choose on of the following: • Continue processing. • Stop processing the log stream all together. • Attempt to get the problem rectified if possible, then attempt to re-read the log data. For an IXGDELET request, the block identifier of the first accessible block toward the youngest data in the log stream is returned in the ANSAA_GAPS_NEXT_BLKID field in the answer area mapped by the IXGANSAA macro. If appropriate, re-issue the IXGDELET request using this block identifier.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0837	**Equate Symbol**: IxgRsnCodeBadTimestamp **Explanation**: Program error. The storage area specified by TIMESTAMP cannot be accessed. **Action**: Ensure that the storage area is accessible to the system logger service for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx083B	**Equate Symbol**: IxgRsncodeBadBTokenStor **Explanation**: Program error. The storage area specified by BROWSETOKEN cannot be accessed. **Action**: Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx083D	**Equate Symbol**: IxgRsnCodeBadECBStor **Explanation**: Program error. The ECB storage area was not accessible to the system logger. **Action**: Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller's home address space and in the same key as the caller.
08	xxxx083F	**Equate Symbol**: IxgRsnCodeTestartError **Explanation**: System error. An unexpected error was encountered while attempting to validate the buffer ALET. **Action**: See ANSAA_DIAG1 in the answer area mapped by the IXGANSAA macro for the return code from the TESTART system service.
08	xxxx0841	**Equate Symbol**: IxgRsnCodeBadBufferAlet **Explanation**: Program error. The buffer ALET specified is not zero and does not represent a valid entry on the caller's dispatchable unit access list (DUAL). See the ANSAA_DIAG1 field of the answer area, mapped by the IXGANSAA macro, for the return code from the TESTART system service. **Action**: Ensure that the correct ALET was specified. If not, provide the correct ALET. Otherwise, add the correct ALET to dispatchable unit access list (DUAL).

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0845	**Equate Symbol**: IxgRsnCodeInvalidFunc **Explanation**: System error. One of 2 problems was detected. 1. The parameter list for this service contains an unrecognizable function code. The parameter list storage may have been overlayed. 2. The IXGBRWSE START is rejected because either: • A: An unauthorized caller attempted to start a session when 100 or more browse sessions already exist for this connection. Or, • B: An unauthorized caller attempted to start a session when 20 or more browse sessions already exist that show no recent activity. (An unauthorized caller is a caller whose PSW Key is >= 8 and that is not in supervisor state). For Case 2: DIAG1 in the Answer Area will contain 1 if 'A' is the case, and 2 if 'B' is the case. DIAG2 will contain the number of browse sessions that was exceeded. **Action**: Fix the problem and then re-issue the request. It may be necessary to terminate some Browse sessions that are not being used.
08	xxxx0846	**Equate Symbol**: IxgRsnCodeEmptyStream **Explanation**: Environment error. The log stream is empty. **Action**: Wait for data to be written to the log stream before browsing for data.
08	xxxx0847	**Equate Symbol**: IxgRsnCodeEOFDelete **Explanation**: Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on the direction of the read) was deleted from the log stream. **Action**: Determine whether this is an acceptable condition for your application. If so, ignore this condition. If not, provide serialization on the log stream or some other installation protocol to prevent deletes from being performed by other applications during a browse session.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0848	**Equate Symbol**: IxgRsnCodeEndReached **Explanation**: Environment error. The request failed and no log data is returned. For a READCURSOR request, the end of the log stream has been reached in the direction of the read. If the SEARCH parameter was specified on a READBLOCK request, the time stamp is greater than any block in the log stream. **Action**: For the READCURSOR case, no more data exists in the log stream in the direction of the read. You can choose to stop reading, wait for more data to be written, or change the direction of the read. In the case where the SEARCH parameter was provided, ensure that the time stamp is less than or equal to the highest time stamp of a log block in the log stream.
08	xxxx0849	**Equate Symbol**: IxgRsnCodeBadBuffkey **Explanation**: Program error. The buffer key specified on the BUFFKEY parameter specifies an invalid key. Either the key is greater than 15 or the program is running in problem state and the specified key is not the same key as the PSW key at the time the system logger service was issued. **Action**: For problem state programs, either do not specify the BUFFKEY parameter or else specify the same key as the PSW key at the time the system logger service was issued. For supervisor state programs, specify a valid storage key (0 <= key <= 15).
08	xxxx084A	**Equate Symbol**: IxgRsnCodeEOFGap **Explanation**: Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on the direction of the read) was unreadable. This condition may be caused by either an I/O error while trying to read a log data set, or a log data set deleted without using logger interfaces. **Action**: The action necessary is completely up to the application depending on how critical your data is. You can do one of the following: • Accept this condition and continue reading. • Stop processing the log all together. • Attempt to get the problem rectified, if possible, and then attempt to re-issue the request.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx084B	**Equate Symbol**: IxgRsncodeLossOfDataGap **Explanation**: Environment error. The requested log data referenced a section of the log stream where log data is permanently missing. This condition occurs when a system or coupling facility is in recovery due to a failure, but not all of the log data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
08	xxxx084D	**Equate Symbol**: IxgRsnCodeLossOfDataEOF **Explanation**: Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on direction of the read) was permanently lost. This condition occurs when a system or coupling facility is in recovery due to a failure, but not all of the log data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
08	xxxx0852	**Equate Symbol**: IxgRsnCodeBadBlkSizeStor **Explanation**: Program error. The storage area specified on the BLKSIZE parameter cannot be accessed. **Action**: Ensure that the storage area is accessible to system logger for the duration of the request.
08	xxxx085F	**Equate Symbol**: IxgRsnPercToRequestor **Explanation**: Environment error. Percolation to the service requestor's task occurred because of an abend during system logger processing. Retry was not allowed. **Action**: Issue the request again. If the problem persists, contact the IBM Support Center.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0861	**Equate Symbol**: IxgRsnCodeRebuildInProgress **Explanation**: Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0862	**Equate Symbol**: IxgRsnCodeXESPurge **Explanation**: Environment error. An cross-system extended services (XES) request has been purged due to re-build processing. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0863	**Equate Symbol**: IxgRsnCodeStructureFailed **Explanation**: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0864	**Equate Symbol**: IxgRsnCodeNoConnectivity **Explanation**: Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to re-build the log stream in another coupling facility or the log stream will be disconnected. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available. • The log stream has been disconnected from this system. If a re-build initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues.
08	xxxx0890	**Equate Symbol**: IxgRsnCodeAddrSpaceNotAvail **Explanation**: System error. The system logger address space failed and is not available. **Action**: Do not issue system logger requests.
08	xxxx0891	**Equate Symbol**: IxgRsnCodeAddrSpaceInitializing **Explanation**: System error. The system logger address space is not available because it is IPLing. **Action**: Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-connect to the log stream, then re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services.
08	xxxx08D0	**Equate Symbol**: IxgRsnCodeProblemState **Explanation**: Environment error. The request was rejected because of one of the following: • The request was issued in SRB mode while the requestor was in problem program state. • The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key. **Action**: Change the invoking environment to supervisor state.

Table 129. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx08D1	**Equate Symbol**: IxgRsnCodeProgramKey **Explanation**: Environment error. The request was rejected because of one of the following: • The request was issued in SRB mode while the requestor was in problem program key (key 8-F). • The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key. **Action**: Change the invoking environment to a system key (key 0-7).
08	xxxx08D2	**Equate Symbol**: IxgRsnCodeNoCompleteExit **Explanation**: Program error. MODE=SYNCEXIT was specified, but the connection request did not identify a complete exit. **Action**: Either change this request to a different MODE option, or reconnect to the log stream with a complete exit on the COMPLETEXIT parameter.
08	xxxx08D3	**Equate Symbol**: IxgRsnCodeFuncNotSupported **Explanation**: Environment error. The options specified on the IXGBRWSE request are not supported on this system/ maintenance level of system logger. **Action**: Either install the level of system logger that provides the support for the requested function, or do not specify options that are not supported at this level.
0C	xxxx0000	**Equate Symbol**: IxgRetCodeCompError **Explanation**: User or System error. One of the following occurred: • You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space. • System logger component error occurred. **Action**: If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSAA) and any dumps or LOGREC entries from system logger.

Examples

Example 1

Issue IXGBRWSE REQUEST=START to start a browse session, starting the browse cursor at the log block with the specified local time.

```
          IXGBRWSE REQUEST=START,                                      X
               STREAMTOKEN=TOKEN,                                      X
               SEARCH=SRCHTIME,                                        X
               GMT=NO,                                                 X
```

```
               BROWSETOKEN=BRSTOKEN,                                  X
               MODE=SYNC,                                             X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
TOKEN    DS    CL16                   stream token from connect
SRCHTIME DS    2F                     local search time in stck format
BRSTOKEN DS    CL4                    returned browse token
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
RETCODE  DS    F                      return code
RSNCODE  DS    F                      reason code
DATAREA  DSECT
         IXGANSAA LIST=YES            answer area
```

Example 2

Issue IXGBRWSE REQUEST=READCURSOR to read the next consecutive log block in the specified direction. In this example, the default of MULTIBLOCK=NO has been taken.

```
          IXGBRWSE REQUEST=READCURSOR,                                X
               STREAMTOKEN=TOKEN,                                     X
               BUFFER=BUFF,                                           X
               BUFFLEN=BUFFLEN,                                       X
               BUFFALET=ALET,                                         X
               BLKSIZE=BLKSIZE,                                       X
               DIRECTION=OLDTOYOUNG,                                  X
               RETBLOCKID=RETBLK,                                     X
               TIMESTAMP=TIMESTMP,                                    X
               BROWSETOKEN=BRSTOKEN,                                  X
               MODE=SYNC,                                             X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
BUFFLEN  DC    F'200'                 buffer length
TOKEN    DS    CL16                   stream token from connect
BRSTOKEN DS    CL4                    returned browse token
BUFF     DS    CL200                  buffer where data will be put
ALET     DC    F'1'                   buffer alet in secondary
BLKSIZE  DS    F                      block size of buffer
RETBLK   DS    CL8                    return block id
TIMESTMP DS    CL16                   returned time stamp stck format
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
RETCODE  DS    F                      return code
RSNCODE  DS    F                      reason code
DATAREA  DSECT
         IXGANSAA LIST=YES            answer area
```

Example 3

Issue IXGBRWSE REQUEST=READBLOCK to read a log block selected by block identifier.

```
          IXGBRWSE REQUEST=READBLOCK,                                 X
               STREAMTOKEN=TOKEN,                                     X
               BLOCKID=BLKID,                                         X
               BUFFER=BUFF,                                           X
               BUFFLEN=BUFFLEN,                                       X
               BUFFALET=ALET,                                         X
               BLKSIZE=BLKSIZE,                                       X
               RETBLOCKID=RETBLK,                                     X
               TIMESTAMP=TIMESTMP,                                    X
               BROWSETOKEN=BRSTOKEN,                                  X
               MODE=SYNC,                                             X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
BUFFLEN  DC    F'200'                 buffer length
```

```
TOKEN    DS    CL16                stream token from connect
BRSTOKEN DS    CL4                 returned browse token
BUFF     DS    CL200               buffer where data will be put
ALET     DS    F'1'                buffer alet in secondary
BLKSIZE  DS    F                   block size of buffer
RETBLK   DS    CL8                 return block id
BLKID    DS    CL8                 specific block id to browse
TIMESTMP DS    CL16                returned time stamp stck format
ANSAREA  DS    CL(ANSAA_LEN)       answer area for log requests
RETCODE  DS    F                   return code
RSNCODE  DS    F                   reason code
DATAREA  DSECT
         IXGANSAA LIST=YES         answer area
```

Example 4

Issue IXGBRWSE REQUEST=RESET to reset the cursor at the youngest block in the log stream.

```
          IXGBRWSE REQUEST=RESET,                                  X
               STREAMTOKEN=TOKEN,                                  X
               POSITION=YOUNGEST,                                  X
               BROWSETOKEN=BRSTOKEN,                               X
               MODE=SYNC,                                          X
               ANSAREA=ANSAREA,                                    X
               ANSLEN=ANSLEN,                                      X
               RSNCODE=RSNCODE,                                    X
               MF=S,                                               X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)        length of logger's answer area
TOKEN    DS    CL16                stream token from connect
BRSTOKEN DS    CL4                 returned browse token
ANSAREA  DS    CL(ANSAA_LEN)       answer area for log requests
RETCODE  DS    F                   return code
RSNCODE  DS    F                   reason code
DATAREA  DSECT
         IXGANSAA LIST=YES         answer area
```

Example 5

Issue IXGBRWSE REQUEST=END to end a browse session.

```
          IXGBRWSE REQUEST=END,                                    X
               STREAMTOKEN=TOKEN,                                  X
               BROWSETOKEN=BRSTOKEN,                               X
               MODE=SYNC,                                          X
               ANSAREA=ANSAREA,                                    X
               ANSLEN=ANSLEN,                                      X
               RSNCODE=RSNCODE,                                    X
               MF=S,                                               X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)        length of logger's answer area
TOKEN    DS    CL16                stream token from connect
BRSTOKEN DS    CL4                 browse token from browse start
ANSAREA  DS    CL(ANSAA_LEN)       answer area for log requests
RETCODE  DS    F                   return code
RSNCODE  DS    F                   reason code
DATAREA  DSECT
         IXGANSAA LIST=YES         answer area
```

Example 6

Issue IXGBRWSE REQUEST=END to end a browse session asynchronously, if synchronous processing is not possible.

```
          IXGBRWSE REQUEST=END,                                    X
               STREAMTOKEN=TOKEN,                                  X
               BROWSETOKEN=BRSTOKEN,                               X
               MODE=SYNCECB,                                       X
               ECB=ANECB,                                          X
               ANSAREA=ANSAREA,                                    X
               ANSLEN=ANSLEN,                                      X
               RSNCODE=RSNCODE,                                    X
               MF=S,                                               X
```

```
               RETCODE=RETCODE
*********************************************************************
*        if rsncode = '00000401'X then wait on
*        the ecb ANECB.
*********************************************************************
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
TOKEN    DS    CL16                   stream token from connect
BRSTOKEN DS    CL4                    browse token from browse start
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
ANECB    DS    F                      ecb on which to wait
RETCODE  DS    F                      return code
RSNCODE  DS    F                      reason code
DATAREA  DSECT
         IXGANSAA LIST=YES            answer area
```

Example 7

Issue IXGBRWSE REQUEST=END using registers.

```
          LA   R6,TOKEN               place stream token in reg 6
          IXGBRWSE REQUEST=END,                                      X
               STREAMTOKEN=(6),                                      X
               BROWSETOKEN=BRSTOKEN,                                 X
               MODE=SYNC,                                            X
               ANSAREA=ANSAREA,                                      X
               ANSLEN=ANSLEN,                                        X
               RSNCODE=RSNCODE,                                      X
               MF=S,                                                 X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
TOKEN    DS    CL16                   stream token from connect
BRSTOKEN DS    CL4                    browse token from browse start
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
RETCODE  DS    F                      return code
RSNCODE  DS    F                      reason code
DATAREA  DSECT
         IXGANSAA LIST=YES            answer area
R6       EQU   6
```

Chapter 144. IXGCONN — Connect/disconnect to log stream

Description

Use the IXGCONN macro to connect a program to a specific log stream or disconnect a program from a specific log stream.

IXGCONN returns a unique connection identifier called a stream token on completion of the IXGCONN REQUEST=CONNECT request. Subsequent logger services use the stream token to identify the connection. If multiple applications connect to the same log stream, the log blocks written from the different applications are merged.

The IXGCONN connect service can be used in the following ways:

- Once a program has connected to a log stream, any application running in the same address space shares the connect status and may share the same stream token to issue other logger services. Any program in the address space can disconnect the entire address space from the log stream by issuing the IXGCONN REQUEST=DISCONNECT service.
- Multiple programs in a single address space can issue IXGCONN REQUEST=CONNECT individually to connect to the same log stream and receive separate stream tokens. Each program must disconnect from the log stream individually.
- Multiple address spaces on one or more MVS systems may connect to a single log stream, but each one must issue IXGCONN individually to connect and then disconnect from the log stream. Each one receives a unique stream token; address spaces cannot share a stream token.

  Note that a DASD-only log stream is single-system in scope. This means that only one system may connect to a DASD-only log stream, although there can be multiple connections from that one system.

The parameter descriptions indicate parameters that can only be used in supervisor state, PSW key zero. All others can be used in problem or supervisor state with any PSW key.

For information about using the system logger services and the IXGCONN request, see *z/OS MVS Programming: Assembler Services Guide* which includes information about related macros IXGBRWSE, IXGDELET, IXGWRITE, IXGINVNT, and IXGQUERY.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN, any SASN
AMODE:	31-bit or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks may be held.
Control parameters:	None.

Programming requirements

- The parameter list for this service must be addressable in the caller's primary address space.
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include mapping macro IXGANSAA in your program. This macro shows the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- If you use IXGCONN REQUEST=CONNECT,...,MF=(E,parmlist,NOCHECK) with either the STREAMTOKEN=xxxx or the USERDATA=yyyy keyword, the following procedure must be followed. When the processing is complete, move the STREAMTOKEN or USERDATA values from the parameter list specified on MF= to your own storage.
- Each task that issues IXGCONN REQUEST=CONNECT to connect to a log stream must later issue IXGCONN REQUEST=DISCONNECT to disconnect from the log stream. When a task disconnects from the log stream, the log stream token that identified the connection expires. Any requests that use the log stream token after the disconnect are rejected with reason code X'82D'.
- If a task that issued the IXGCONN REQUEST=CONNECT request ends before issuing a disconnect request, system logger automatically disconnects the task from the log stream. This means that the unique log stream connection identifier, or the STREAMTOKEN, is no longer valid. The application receives an expired log stream token error response with reason code X'82D', if this application continues to use the same STREAMTOKEN after the task has been disconnected on subsequent logger service requests.
- Any job step task (JST) terminates within the address space that has a connection to the log stream. System logger treats any job step task termination in a manner similar to an address space termination. That is, all log stream connections are disconnected and logger associations are terminated with the address space.

  If this condition occurs and there remains an expected use of a log stream, then a new log stream connection will be required.

Restrictions

- All storage areas specified in this service must be in the same storage key as the caller's storage key and must exist in the caller's primary address space.
- The caller cannot have an EUT FRR established.
- If the System Authorization Facility (SAF) is available, the system performs SAF authorization checks on all IXGCONN REQUEST=CONNECT requests in order to protect the integrity of data in a log stream.

  To connect successfully to a log stream, the caller must have SAF authorization that matches the authorization required for the log stream:

  – To connect to a log stream with an authorization level of READ, the caller must have read (or higher level) access to RESOURCE(*log_stream_name*) in SAF class CLASS(LOGSTRM).

  – To connect to a log stream with an authorization level of WRITE, profiles covering two separate resources are used to determine if access is allowed and whether full access to log stream services can be used or just limited log stream access (only IXGWRITE and disconnect requests) can be used for the connection. Refer to each log stream exploiter's planning and/or guidance documentation on their log stream access requirements. Grant permission to SAF profile covering the (WRITE_ONLY_*log-stream-name*) resource only when deemed appropriate for the log exploiter.

    - For full service WRITE log stream access, the caller must have update (or higher level) access to RESOURCE(*log_stream_name*) in SAF class CLASS(LOGSTRM). When this level of access is granted, all log stream access services can be used with the log stream token established on this log stream connection.
    - For limited service WRITE log stream access, the caller must have (at least) update access to RESOURCE(WRITE_ONLY_*log-stream-name*) in SAF CLASS(LOGSTRM). When this level of access is

granted, only IXGWRITE and IXGCONN (to disconnect) requests can be used with the log stream token established on this log stream connection.

- When there are no profiles defined covering resources (*log-stream-name*) or (WRITE_ONLY_*log-stream-name*) in SAF CLASS(LOGSTRM), then the AUTH=WRITE connection is permitted as full service access to the log stream.
- When just the profile covering resource (*log-stream-name*) is defined, meaning no profile covering (WRITE_ONLY_*log-stream-name*), then the connection is permitted according to the (*log-stream-name*) profile access. Access to the log stream will either be full service access or connection is not allowed (denied).
- When just the profile covering resource (WRITE_ONLY_*log-stream-name*) is defined, meaning no profile covering (*log-stream-name*), then the connection is permitted based on the (WRITE_ONLY_*log-steram-name*) profile access. Access to the log stream will either be limited service log stream access (can only write log data and disconnect) or the connection is not allowed (denied).
- When profiles covering both resources (WRITE_ONLY_*log-stream-name*) and (*log-stream-name*) are defined, then connection is permitted based on the following combinations:
  - when access is permitted via both profiles, then access to the log stream will be full service access.
  - when access is permitted via (*log-stream-name*) profile, but is denied via (WRITE_ONLY_*log-stream-name*) profile, then access to the log stream will be full service access.
  - when access is permitted via (WRITE_ONLY_*log-stream-name*) profile, but is denied via (*log-stream-name*) profile, then access to the log stream will be limited service (can only write log data and disconnect).
  - when access is not permitted (denied) via both profiles, then the connection to the log stream will not be allowed (denied).

  When permission is not allowed for the profiles covering the resources noted above, system logger will provide an error return, with return code 8, reason code `X'080D'` (IxgRsnCodeNoSAFAuth) on the log stream connection attempt.

If SAF is not available or if CLASS(LOGSTRM) is not defined to SAF, no security checking is performed. In that case, the caller is connected to the log stream with the requested or default AUTH parameter value.

- There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.

Input register information

Before issuing the IXGCONN macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if register 15 contains a non-zero return code

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

Some messages and WTORs can be issued to delay or fail the IXGCONN Request. These messages and WTORs are issued when Logger is waiting for other system services. The following messages may need to be replied to, or other action taken:

- IXG054A - *cdstype* CDS not yet made available for Logger's use.
- IXG254I - SMS is not yet active
- IXG115A - Log stream recovery not making progress trying to move recovered log data to secondary (offload) data sets.

  See the topic on IXG Messages in *z/OS MVS System Messages, Vol 10 (IXC-IZP)* for more information about IXG messages.

Syntax

The standard form of the IXGCONN macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IXGCONN.
IXGCONN	
␣	One or more blanks must follow IXGCONN.
	Valid parameters (Required parameters are underlined.)
REQUEST=CONNECT	All parameters are valid.
REQUEST=DISCONNECT	STREAMTOKEN, ANSAREA, ANSLEN, USERDATA, RETCODE, RSNCODE, MF
,STREAMNAME=*streamname*	*streamname*: RS-type address or register (2) - (12).

Syntax	Description
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,AUTH=READ	**Default**: AUTH=READ
,AUTH=WRITE	
,STRUCTNAME=*structname*	*structname*: RS-type address or register (2) - (12).
,AVGBUFSIZE=*avgbufsize*	*avgbufsize*: RS-type address or register (2) - (12).
,MAXBUFSIZE=*maxbufsize*	*maxbufsize*: RS-type address or register (2) - (12).
,ELEMENTSIZE=*elementsize*	*elementsize*: RS-type address or register (2) - (12).
,LSVERSION=*lsversion*	*lsversion*: RS-type address or register (2) - (12).
,COMPLETEEXIT=*completeexit*	*completeexit*: RS-type address or register (2) - (12).
,USERDATA=*userdata*	*userdata*: RS-type address or register (2) - (12).
,IMPORTCONNECT=NO	**Default**: IMPORTCONNECT=NO
,IMPORTCONNECT=YES	
,DIAG=NO_DIAG	**Default**: DIAG=NO_DIAG
,DIAG=NO	

Syntax	Description
,DIAG=YES	
,RMNAME=*rmname*	*rmname*: RS-type address or register (2) - (12).
,RMEXIT=*rmexit*	*rmexit*: RS-type address or register (2) - (12).
,RMDATA=*rmdata*	*rmdata*: RS-type address or register (2) - (12).
,RMEVENTS=LBWRITE	
,RMEVENTS=LBDELETE	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=1	
,PLISTVER=2	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default**: MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained as follows:

REQUEST=CONNECT
REQUEST=DISCONNECT

Input parameter specifying whether the program is connecting to or disconnecting from the specified log stream.

When you specify CONNECT, all parameters are valid. Keywords required with connect are: STREAMNAME, STREAMTOKEN, ANSAREA, and ANSLEN.

When you specify DISCONNECT, the following parameters are valid (required parameters are underlined): STREAMTOKEN, ANSAREA, ANSLEN, USERDATA, RETCODE, RSNCODE, and MF.

,STREAMNAME=*streamname*

Specifies the 26-byte field (or register) containing the name of the log stream to which a program is connecting. You must use the name you defined for the log stream in the LOGR, LOGRY or LOGRZ couple data set policy that is active on the system where the log stream is being connected. See the IXGINVNT macro for information on the syntax of log stream names in the active system logger policy.

,STREAMTOKEN=*streamtoken*

Specifies the 16-byte token uniquely identifying the program's connection to the log stream.

When specified with REQUEST=CONNECT, STREAMTOKEN is an output parameter where IXGCONN places the log stream token when the macro completes successfully.

When specified with REQUEST=DISCONNECT or other logger services, STREAMTOKEN is an input parameter where you specify the log stream token returned at connection.

,ANSAREA=*ansarea*

Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*

Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,AUTH=READ
,AUTH=WRITE

Specifies whether the caller has write or read access to the specified log stream.

If you specify AUTH=READ when connecting to a log stream, the program must also have read access authority to SAF resource(*logstream_name*) in CLASS(LOGSTRM) for the specified log stream. You can then issue only the IXGBRWSE and IXGQUERY requests against the log stream.

If you specify AUTH=WRITE when connecting to a log stream, the program must also have write access authority to SAF resource(*logstream_name*) in CLASS(LOGSTRM) for the specified log stream. You can then issue any system logger request against the log stream.

AUTH=WRITE specifies that, based on the defined security profiles and allowed access, either all log stream access services (such as IXGBRWSE, IXGDELET, IXGWRITE and others) or just limited log stream access requests (just writing log data and disconnecting from the log stream) can be issued for the log stream.

When the request results in a log stream connection (rc=0 or rc=4 conditions), then the output answer area flag field Ansaa_WriteConnectLimitedAccess (see mapping macro IxgAnsaa) indicates whether full log stream access services are allowed or whether only limited log stream service access is allowed.

When the indicator Ansaa_WriteConnectLimitedAccess is set off `'0'b` then full log stream access services area is allowed. This means that all the log stream access services using the log stream connection token returned in the STREAMTOKEN parameter and that are given the context of the other connection parameters, can be issued.

When the indicator Ansaa_WriteConnectLimitedAccess is set on '1'b along with indicator Ansaa_WriteOnlyAccess being set on '1'b, then the log stream access services area limited to writing log data and disconnecting from the log stream (IXGWRITE and IXGCONN requests). No other log stream access services are allowed for this limited access connection type

,STRUCTNAME=*structname*
Specifies the name or address (using a register) of a 16-byte output field where IXGCONN REQUEST=CONNECT will return the name of the coupling facility structure that the log stream is connected to. The name comes from the LOGR policy.

If you are connecting to a DASD-only log stream, this field will contain binary zeros. In addition, flag Ansaa_DasdOnlyLogStream in macro IXGANSAA will be set on for a DASD-only log stream.

,MAXBUFSIZE=*maxbufsize*
Specifies the name or address (using a register) of a 4-byte output field where IXGCONN returns the size, in bytes, of the largest log block that can be written to this log stream.

MAXBUFSIZE is defined in the active system logger couple data set policy.

,AVGBUFSIZE=*avgbufsize*
Specifies the name or address (using a register) of a 4-byte output field where IXGCONN returns the average size, in bytes, of individual log blocks that can be written to the coupling facility structure associated with this log stream.

AVGBUFSIZE is defined in the active system logger couple data set policy.

- If you are using an active system logger couple data set for a coupling facility log stream, this value shows the initial setting used to determine the element-to-entry ratio. System logger monitors structure usage and adjusts the average buffer size dynamically, but the AVGBUFSIZE value returned by IXGCONN will always reflect the original setting rather than the actual value in use by system logger at any given time.
- If you are connecting to a DASD-only log stream, this field will contain binary zeros. In addition, flag Ansaa_DasdOnlyLogStream in macro IXGANSAA will be set on for a DASD-only log stream.

,ELEMENTSIZE=*elementsize*
Specifies the name or address (using a register) of a 4-byte output field where IXGCONN returns the size of the elements that system logger will break the log blocks into to write them to the coupling facility associated with this log stream.

If you are connecting to a DASD-only log stream, this field will contain binary zeros. In addition, flag Ansaa_DasdOnlyLogStream in macro IXGANSAA will be set on for a DASD-only log stream.

,LSVERSION=*lsversion*
Specifies the name or address (using a register) of a 64-bit output field where IXGCONN returns the version of the log stream the program is connecting to.

The log stream version is a UTC timestamp that uniquely identifies the instance of the log stream definition. A program can use the log stream version to see if a log stream definition has been deleted and redefined since the last connect to a log stream.

For example, assume you connect to log stream LS1 and IXGCONN returns a log steam version of X'AA00000000000000', which the program saves. On a subsequent connection to log stream LS1, IXGCONN returns a different log stream version, which indicates that the definition for log stream LS1 in the active system logger couple data set policy has been deleted and redefined since the last connection.

,COMPLETEEXIT=*completeexit*
Specifies the name or address (using a register) of a user exit called a complete exit. Use this parameter to specify a complete exit for the caller. The complete exit gets control when the system processes IXGBRWSE, IXGDELET, or IXGWRITE requests that specify MODE=SYNCEXIT asynchronously. The complete exit receives control in SRB mode, supervisor state, key 0, enabled, and unlocked.

If you specify a name for this parameter, it must be the name of an entry point addressable in the invoking load module. For example, the name can be a routine name that exits in the invoking module or a CSECT link-edited into the invoking load module.

The caller must ensure that the complete exit routine is loaded into either private storage in the connector's primary address space or common storage. The exit must remain loaded in storage until all asynchronous requests that have specified SYNCEXIT with the log stream token returned by this connect request have completed. Even if the log stream is disconnected, you cannot assume that all SYNCEXIT requests have completed.

For more information on coding a complete exit and the environment where the complete exit runs, see *z/OS MVS Programming: Assembler Services Guide*.

,USERDATA=*userdata*

Specifies a 64-byte input/output field containing a user data area.

When specified with REQUEST=CONNECT, USERDATA is an output parameter where IXGCONN returns the user data specified for this log stream.

When specified with REQUEST=DISCONNECT, USERDATA is an input parameter where you can specify or update the user data the user data for the specified log stream. You can only specify or change the user data for a log stream on a disconnect request.

,IMPORTCONNECT=NO
,IMPORTCONNECT=YES

Specifies whether the connection is for writing or importing log data to a log stream. You must specify AUTH=WRITE to use the IMPORTCONNECT parameter.

If you specify IMPORTCONNECT=YES, this connection will be used for importing data to a log stream. Importing log data means using the IXGIMPRT service to copy data from one log stream to another, maintaining the same log block identifier and UTC time stamp. IXGWRITE requests are not valid with IMPORTCONNECT=YES. You can have only one IMPORTCONNECT=YES connection active for a log stream in the sysplex.

If you specify IMPORTCONNECT=NO, which is the default, the connect request is a write connection. In a write connection, only IXGWRITE requests can be issued against the log stream, IXGIMPRT requests will be rejected.

You can have multiple write connects to a log stream, provided there are no import connections. If you have a write connect established against a log stream, a subsequent import connection will be rejected. You cannot, in other words, issue both IXGIMPRT and IXGWRITE requests against a single log stream.

This option is not allowed when the AUTH=WRITE connection would have otherwise been established as a limited log stream access, given the SAF profiles defined for the log stream access. The request will fail with a return code 8, reason code X'08D6' (IxgRsnCodeConnTypeNotAllowed). The answer area indicators Ansaa_WriteConnectLimitedAccess and Ansaa_WriteOnlyAccess will be set on ('1'b) for this error condition.

,RMNAME=*rmname*

Specifies the name (or address in a register) of the 8-byte input field containing the name of the resource manager program connecting to the log stream. The resource manager name specified on the IXGCONN request must be the same as the one associated with the log stream in the log stream definition in the LOGR policy. The application must run in supervisor state, key 0-7 to use this parameter.

The RMNAME parameter is specified only by the resource manager at connect time, to tell system logger that it is connecting to a log stream. Other connections to a resource manager managed log stream do not have to specify RMNAME. Note that a resource manager can only connect to one log stream per system.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

,RMEXIT=*rmexit*
Specifies the name (or address in a register) of the input field containing the address of user exit for the resource manager. The application must run in supervisor state, key 0-7 to use this parameter.

RMEXIT is required with the RMNAME parameter, even though use of a resource manager exit is optional. The exit is called only if the resource manager monitors write and/or delete events as selected on the RMEVENT parameter.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

RMEXIT is required when you specify RMNAME.

,RMDATA=*rmdata*
Specifies the name (or address in a register) of the 8-byte input field containing the data for the user exit. The application must run in supervisor state, key 0-7 to use this parameter.

RMDATA is required with the RMNAME parameter.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

,RMEVENT=LBWRITE
,RMEVENT=LBDELETE
Input parameter specifying the events that you want to trigger the resource manager user exit. RMEVENT is required with the RMNAME parameter. You can specify RMEVENTS=LBWRITE, RMEVENTS=LBDELETE, or RMEVENTS=(LBWRITE,LBDELETE). The application must run in supervisor state, key 0-7 to use this parameter.

If you specify RMEVENT=LBWRITE, successful write requests to the log stream will trigger the resource manager user exit.

If you specify RMEVENT=LBDELETE, successful delete requests to the log stream will trigger the resource manager user exit.

The active primary LOGR couple data set must be formatted at the z/OS level to use this parameter.

,DIAG=NO_DIAG
,DIAG=NO
,DIAG=YES
Specifies whether Logger should provide additional diagnostics as specified on the logstream definition DIAG parameter. This indication is used over the span of this connectoin. Refer to the DIAG keyword on the IXGINVNT, IXGBRWSE, and IXGDELET macro services.

If you specify DIAG=NO_DIAG, which is the default, then Logger will not provide the additional diagnostics as specified on the logstream definition DIAG parameter, unless another Logger service, for example, IXGBRWSE, specifically requests the additional diagnostics.

If you specify DIAG=NO, the Logger will not provide the additional diagnostics as specified on the logstream definition DIAG parameter, regardless of other Logger service specifications.

If you specify DIAG=YES, then Logger will provide additional diagnostics as specified on the logstream definition DIAG parameter, unless another Logger service, for example, IXGDELET, specifically requests not to provide the additional diagnostics.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=1
,PLISTVER=2
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.

- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **1**, which supports all parameters except those specifically referenced in higher versions.
- **2**, which supports both the following parameters and parameters from version 1:
  - COMPLETEEXIT
  - IMPORTCONNECT
  - LSVERSION
  - RMDATA
  - RMEVENTS
  - RMEXIT
  - RMNAME

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 1 or 2

,RETCODE=*retcode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*
Specifies a name (or address in a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When IXGCONN macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

The IXGCON mapping macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

00
IXGRETCODEOK - Service completes successfully.

04
IXGRETCODEWARNING - Service completes with a warning.

08
IXGRETCODEERROR - Service does not complete.

0C
IXGRETCODECOMPERROR - Service does not complete.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 130. Return and reason codes for the IXGCONN macro

Return code	Reason code	Meaning and zction
00	xxxx0000	**Equate symbol**: IxgRsnCodeOk **Explanation**: Request processed successfully.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
04	xxxx0404	**Equate symbol**: IxgRsnCodeDisconnectInProgress **Explanation**: Environment error. The disconnect request is being completed asynchronously. The application has been disconnected from the log stream and the stream token is no longer valid. **Action**: The log stream cannot be deleted until the asynchronous portion of the disconnect processing completes.
04	xxxx0406	**Equate symbol**: IxgRsnCodeConnectRebuild **Explanation**: Environment error. The connect request was successful, but the log stream is temporarily unavailable because a coupling facility structure rebuild is in progress. **Action**: Listen to the ENF signal 48, which will indicate either that the log stream is available because the rebuild completed successfully or that the log stream is not available because the rebuild failed. In the meantime, do not attempt to issue system logger services against the log stream.
04	xxxx0407	**Equate symbol**: IxgRsnCodeConnPossibleLossOfData **Explanation**: Environment error. The request was successful, but there may be log blocks permanently missing between this log block and the one previously returned. This condition occurs when a system or coupling facility fails and not all of the data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
04	xxxx0408	**Equate symbol**: IxgRsnCodeDsDirectoryFullWarning **Explanation**: Environment error. The request was successful, but the DASD data set directory for the log stream is now full. system logger cannot offload any further data to DASD. system logger will continue to process IXGWRITE requests only until the coupling facility structure space for this log stream is full. **Action**: Either delete data from the log stream to free up space in the data set directory or disconnect from the log stream.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
04	xxxx0409	**Equate symbol**: IxgRsnCodeWowWarning **Explanation**: Environment error. The request was successful, but an error condition was detected during a previous offload of data. system logger might not be able to offload further data. system logger will continue to process IXGWRITE requests only until the interim storage for the log stream is filled. (Interim storage is the coupling facility for a coupling facility log stream and local storage buffers for a DASD-only log stream.) **Action**: Do not issue any further requests for this log stream and disconnect. Connect to another log stream. Check the system log for message IXG301I to determine the cause of the error. If you cannot fix the error, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
08	xxxx0801	**Equate symbol**: IxgRsnCodeBadParmlist **Explanation**: Program error. The parameter list could not be accessed. **Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx0802	**Equate symbol**: IxgRsnCodeXESError **Explanation**: System error. A severe cross-system extended services (XES) error has occurred. **Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.
08	xxxx0806	**Equate symbol**: IxgRsnCodeBadStmToken **Explanation**: Program error. The stream token was not valid. **Action**: Make sure that the stream token specified is valid.
08	xxxx0808	**Equate symbol**: IxgRsnCodeEIOError **Explanation**: System error. A severe log data set I/O error has occurred. **Action**: Contact the IBM Support Center. Provide the return and reason code.
08	xxxx080A	**Equate symbol**: IxgRsnCodeRequestLocked **Explanation**: Program error. The program issuing the request is holding a lock. **Action**: Ensure that the program issuing the request is not holding a lock.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx080B	**Equate symbol**: IxgRsnCodeNoStream **Explanation**: Program error. The log stream name specified has not been defined in the active system logger couple data set policy. **Action**: Ensure that the required log stream name has been defined in the active system logger couple data set policy. If the definition appears to be correct, ensure that the application is passing the correct log stream name to the service.
08	xxxx080C	**Equate symbol**: IxgRsnCodeStagingAllocError **Explanation**: Environment error. The system encountered a severe dynamic allocation error with the staging data set. ANSAA_DIAG2 of the answer area contains either the dynamic allocation error code, SMS reason code, or media manager reason code. For more information about the error, check for either message IXG251I, which is issued for data set allocation errors, or check for messages issued by the access method. **Action**: If the problem persists, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
08	xxxx080D	**Equate symbol**: IxgRsnCodeNoSAFAuth **Explanation**: Environment error. The user does not have correct SAF authorization for the request. The caller is not authorized to connect to the log stream or the caller specified AUTH=WRITE when connecting to a log stream with only READ authority. **Action**: IXGCONN returns information about the error in the answer area that is mapped by IXGANSAA. Investigate the meaning of ANSAA_Diag1, ANSAA_Diag2 and ANSAA_Diag4. • ANSAA_Diag1 contains the RACF or installation exit return code from the RACROUTE REQUEST=AUTH macro. • ANSAA_Diag2 contains the RACF or installation exit reason code from the RACROUTE REQUEST=AUTH macro. • ANSAA_Diag4 contains the SAF return code from the RACROUTE REQUEST=AUTH macro. See *z/OS Security Server RACROUTE Macro Reference* for information about the RACROUTE macro. Define the required SAF authorization to allow the requestor to connect to the log stream. If authorization has already been defined, either change the authorization to allow UPDATE access to the log stream or change the application to AUTH=READ.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx0811	**Equate symbol**: IxgRsnCodeBadStrname **Explanation**: Environment error. The structure name specified on the STRUCTNAME parameter is not defined in the CFRM policy. **Action**: Make sure that the structure you want to specify is defined in the CFRM policy.
08	xxxx0812	**Equate symbol**: IxgRsnCodeLogStreamRecoveryFailed **Explanation**: Environment error. The log stream could not be recovered so the connection attempt failed. The system issues message IXG210E and/or IXG211E along with message IXG231I providing further information about the error. **Action**: If the problem persists, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
08	xxxx0813	**Equate symbol**: IxgRsnCodeLogStreamDeleted **Explanation**: Environment error. The request to connect to the specified log stream failed because the log stream is being deleted. **Action**: Redefine the log stream in the active system logger couple data set policy and then reissue the connect request.
08	xxxx0814	**Equate symbol**: IxgRsnCodeNotAvailForIPL **Explanation**: Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization. **Action**: See the explanation for system messages issued during system logger initialization.
08	xxxx0815	**Equate symbol**: IxgRsnCodeNotEnabled **Explanation**: Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails. **Action**: Make sure the program issuing the request is enabled for I/O and external interrupts.
08	xxxx0816	**Equate symbol**: IxgRsnCodeBadAnslen **Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro. **Action**: Reissue the request, specifying an answer area of the required size.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx0819	**Equate symbol**: IxgRsnCodeSRBMode **Explanation**: Program error. The calling program is in SRB mode, but task mode is the required dispatchable unit mode for this system logger service. **Action**: Make sure the calling program is in task mode.
08	xxxx081A	**Equate symbol**: IxgRsnCodeMaxStreamConn & IXGINVNT requests **Explanation**: Environment error. This system has reached the limit for the maximum number of log streams that can be concurrently active. One of the following is true: • The limit of 16,384 concurrently active DASDONLY log streams per system has been reached. For this case, the Answer Area field DIAG1 will contain 16,384. • Either the PRODUCTION or TEST GROUP cannot connect to any more log streams. Message IXG075E or IXG076I is issued. In this case, the Answer Area field DIAG1 will contain the number of structures that are in use for this GROUP. • The TEST GROUP has previously failed and a request has been made to define a logstream with GROUP(TEST). Message IXG074I has been previously issued. In this case, the Answer Area field DIAG1 will contain 0. • A Log stream delete cannot be processed because logger needs to perform an internal connect to the Log stream to complete the delete but no more connections are allowed. **Action**: Your workload need to be planned to either consolidate log streams or balance system activity such that fewer log streams are needed during this time frame.
08	xxxx081B	**Equate symbol**: IxgRsnCodePrimaryNotHome **Explanation**: Program error. The primary address space does not equal the home address space. **Action**: Make sure that the primary address space equals the home address space when issuing this system logger service.
08	xxxx081D	**Equate symbol**: IxgRsnCodeRMNameBadState **Explanation**: Program error. The calling program cannot issue IXGCONN with the RMNAME parameter unless it is in supervisor state and system key. **Action**: Make sure the calling program is in supervisor state.
08	xxxx081E	**Equate symbol**: IxgRsnCodeXESStrNotAuth **Explanation**: Environment Error. The system logger address space does not have access authority to the coupling facility structure associated with the log stream specified. **Action**: Make sure the system logger address space has SAF access to the structure.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx081F	**Equate symbol**: IxgRsnCodeXcdsError **Explanation**: System error. System logger encountered an internal problem while processing the active system logger couple data set. **Action**: Contact the IBM Support Center. Provide the return and reason code and the contents of the answer area (ANSAREA field).
08	xxxx0820	**Equate symbol**: IxgRsnCodeBadModelConn **Explanation**: Program error. The program issued an IXGCONN request to connect to a log stream that was defined as a model in the active system logger couple data set policy. You cannot connect to a model log stream. **Action**: Either change the definition of the specified structure so that it is not a model, or else request connection to a different log stream that is not a model.
08	xxxx082D	**Equate symbol**: IxgRsnCodeExpiredStmToken **Explanation**: Environment error. The stream token is no longer valid because the connector has been disconnected. **Action**: Connect to the log stream again before issuing any functional requests.
08	xxxx082E	**Equate symbol**: IxgRsnCodeNoLogrCDSAvail **Explanation**: Environment error. The request failed because no active primary system logger couple data set (CDS) of the expected type is available to this system. The operator was prompted to either make a couple data set available or to indicate that the current request should be rejected. The operator specified that the current request should be rejected. **Action**: System logger services are unavailable until an active system logger couple data set of the expected type (LOGR, LOGRY, LOGRZ) is made available to this system. For more information about using system logger couple data sets, see "Format the LOGR couple data set and make it available to the sysplex" and "Using LOGRZ or LOGRY couple data sets for a single-system scope within a sysplex" in *z/OS MVS Setting Up a Sysplex*. Once the system logger is available using the couple data set, take the necessary steps to cause the function that issued the logger service to reattempt the request.
08	xxxx0831	**Equate symbol**: IxgRsnCodeBadStreamName **Explanation**: Program error. The log stream name specified on the STREAMNAME parameter is not valid. **Action**: Issue the request again with a valid log stream name on the STREAMNAME parameter.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx083A	**Equate symbol**: IxgRsnCodeRMNameNotAllowed **Explanation**: Program error. The request specified the RMNAME parameter, but the log stream is not defined as having an associated resource manager. **Action**: Either remove the RMNAME keyword from the connection request or update the log stream definition in the system logger inventory.
08	xxxx0843	**Equate symbol**: IxgRsnCodeXcdsReformat **Explanation**: Program error. A couple data set record is not valid. **Action**: Format the system logger couple data set again.
08	xxxx084C	**Equate symbol**: IxgRsnCodeRMAlreadyConnected **Explanation**: Program error. The resource manager is trying to connect to a log stream that it is already connected to. Only one connection specifying RMNAME can be active for a log stream. **Action**: Correct the program so that it does not try to reconnect to the log stream.
08	xxxx084E	**Equate symbol**: IXGRSNCODESTRSACETOOSMALL **Explanation**: Environment error. Structure resources are not available to satisfy the request. All structure resources are allocated as system logger control resources. This condition occurs when the structure resources are consumed by the logstreams connections. **Action**: Increase the size of the structure in the CFRM policy or use the SETXCF ALTER command to dynamically increase the size of the structure.
08	xxxx084F	**Equate symbol**: IxgRsnCodeInvalidRMNameSpecified **Explanation**: Program error. The Resource Manager name specified on the IXGCONN request does not match the RMNAME specified for the log stream in the system logger inventory. **Action**: Either change the IXGCONN request or update the log stream's definition in the system logger inventory.
08	xxxx0850	**Equate symbol**: IXGRSNCODEBADVECTORLEN **Explanation**: Environment error. The connect request was rejected. system logger was unable to locate a vector table in the hardware system area (HSA) that is large enough for the number of log streams associated with it. **Action**: Add storage to the vector storage table and/or retry the connect request later, when storage might be available.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx0851	**Equate symbol**: IXGRSNCODEBADCFLEVEL **Explanation**: Environment error. The connect request was rejected. The operational level of the coupling facility is not sufficient to support logger functions. **Action**: Ensure that the coupling facility operational level for logger structures is at the required level. See *z/OS MVS Setting Up a Sysplex*.
08	xxxx0853	**Equate symbol**: IxgRsnCodeNoCF **Explanation**: Environment error. The connect request was rejected. system logger could not allocate coupling facility structure space because no suitable coupling facility was available. **Action**: Check accompanying message IXG206I for a list of the coupling facilities where space allocation was attempted and the reason why each attempt failed.
08	xxxx0861	**Equate symbol**: IxgRsnCodeRebuildInProgress **Explanation**: Environment error. No requests can be processed for this log stream because a coupling facility structure rebuild is in progress for the structure associated with this log stream. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0862	**Equate symbol**: IxgRsnCodeXESPurge **Explanation**: Environment error. An cross-system extended services (XES) request has been purged due to rebuild processing. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0863	**Equate symbol**: IXGRSNCODESTRUCTUREFAILED **Explanation**: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx0864	**Equate symbol**: IXGRSNCODENOCONNECTIVITY **Explanation**: Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to rebuild the log stream in another coupling facility or the log stream will be disconnected. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available. • The log stream has been disconnected from this system. If a rebuild initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues.
08	xxxx0866	**Equate symbol**: IXGRSNCODESTRUCTUREFULL **Explanation**: Environment error. The coupling facility structure space is full. **Action**: Listen to the ENF signal 48 which will indicate that space is available for the structure after data has been offloaded to DASD.
08	xxxx0890	**Equate symbol**: IXGRSNCODEADDRSPACENOTAVAIL **Explanation**: System error. The system logger address space failed and is not available. **Action**: Do not issue system logger requests.
08	xxxx0891	**Equate symbol**: IXGRSNCODEADDRSPACEINITIALIZING **Explanation**: System error. The system logger address space is not available because it is IPLing. **Action**: Listen for ENF signal 48, which will indicate when the system logger address space is available. Reissue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	xxxx08B0	**Equate symbol**: IXGRSNCODESTRUCTURENOTAVAIL **Explanation**: Environment error. The connect request failed. The structure associated with the log stream is temporarily unavailable because either a rebuild is in progress, a structure dump is in progress, or connections to the structure are being prevented. **Action**: Listen for ENF signal 48, which indicates that a coupling facility is available, and then retry the connect.
08	xxxx08D3	**Equate symbol**: IXGRsnCodeFuncNotSupported **Explanation**: Environment error. An IXGCONN request specified the RMNAME or ImportConnect keyword. However, the logger inventory CDS is downlevel. **Action**: Format the highest level system logger couple data set supported by the systems in the sysplex and make it the active primary system logger CDS.
08	xxxx08D6	**Equate symbol**: IXGRsnCodeConnTypeNotAllowed **Explanation**: Environment error. One of the following occurred: • The connect request specified IMPORTCONNECT=YES, but there is already an active write connection (AUTH=WRITE IMPORTCONNECT=NO) in the sysplex. You cannot have an import connection and a write connection to the same log stream. • The connect request specified AUTH=WRITE IMPORTCONNECT=NO, but there is already an active import connection (IMPORTCONNECT=YES) for the log stream. You cannot have an import connection and a write connection to the same log stream. You can only have one import connection to a log stream. You may have multiple write connections, as long as there is no import connection against a log stream. **Action**: Correct your program and retry the request.
08	xxxx08E2	**Equate symbol**: IxgRsncodeDasdOnlyConnected **Explanation**: Environment error. An attempt to connect to a DASD-only log stream, defined in the active primary sysplex-scope system logger CDS, was rejected on this system because the log stream is already connected on another system in the sysplex. **Action**: Determine which system you want to have a connection to the log stream. If you need this connection, disconnect the first system connection to the log stream and retry this connect request.

Table 130. Return and reason codes for the IXGCONN macro (continued)

Return code	Reason code	Meaning and zction
08	000008E3	**Equate symbol**: IxgRsnCodeLogstreamNotSupported **Explanation**: Environment error. An attempt to connect for the log stream is rejected on this system because the system release level does not support this type of log stream. For example, this system does not support DASD-only log streams, or a log stream attribute such as EHLQ or DUPLEXMODE(DRXRC) cannot be processed on this system release level. **Action**: If you must connect to a DASD-only log stream, make sure you do one of the following: • Update the log stream definition in the LOGR policy to a coupling facility one by specifying a structure name on the definition. • To issue a request for a log stream that has the EHLQ attribute, you must be on a system that is at z/OS Version 1 Release 3 or higher. If you must connect to a log stream with the EHLQ attribute specified, make sure you connect from a system that is at z/OS Version 1 Release3 or higher. If you must connect to a log stream with the DUPLEXMODE(DRXRC) attribute specified, make sure you connect from a system that is at z/OS Version 1 Release 7 or higher.
0C	xxxx0000	**Equate symbol**: IxgRetCodeCompError **Explanation**: User or System error. One of the following occurred: • You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space. • system logger component error occurred. **Action**: If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSAA) and any dumps or LOGREC entries from system logger.

Example 1

Issue IXGCONN REQUEST=CONNECT to connect to a log stream with write authority.

```
          IXGCONN REQUEST=CONNECT,                                      X
               STREAMNAME=STRMNAME,                                     X
               STREAMTOKEN=TOKEN,                                       X
               AUTH=WRITE,                                              X
               ANSAREA=ANSAREA,                                         X
               ANSLEN=ANSLEN,                                           X
               RSNCODE=RSNCODE,                                         X
               MF=S,                                                    X
               RETCODE=RETCODE
STRMNAME  DC    CL26'LOG.STREAM.NAME'  stream name
ANSLEN    DC    A(L'ANSAREA)           length of logger's answer area
TOKEN     DS    CL16                   returned stream token
```

```
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
RETCODE  DS    F                      return code from logger
RSNCODE  DS    F                      reason code from logger
DATAREA  DSECT
         IXGANSAA LIST=YES               answer area
```

Example 2

Issue IXGCONN REQUEST=CONNECT using registers.

```
          LA    R6,STRMNAME             load stream name into reg 6
          IXGCONN REQUEST=CONNECT,                                    X
               STREAMNAME=(6),                                        X
               STREAMTOKEN=TOKEN,                                     X
               AUTH=WRITE,                                            X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
STRMNAME  DC    CL26'LOG.STREAM.NAME'  stream name
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
TOKEN    DS    CL16                   returned stream token
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
RETCODE  DS    F                      return code from logger
RSNCODE  DS    F                      reason code from logger
DATAREA  DSECT
         IXGANSAA LIST=YES            answer area
R6       EQU   6                      set up register 6
```

Example 3

Issue IXGCONN REQUEST=CONNECT as an import connect. This means the connection may issue IXGIMPRT to import data to a log stream.

```
         IXGCONN REQUEST=CONNECT,                                     X
               STREAMNAME=ONAME,                                      X
               STREAMTOKEN=OTOKEN,                                    X
               AUTH=WRITE,                                            X
               IMPORTCONNECT=YES,                                     X
               ANSAREA=XANSAREA,                                      X
               ANSLEN=XANSLEN,                                        X
               RSNCODE=RSCODE

*
ONAME    DS    CL26                    Output Stream name
STOKEN   DS    CL16                    Input Stream token
XANSAREA DS    CL(ANSAA_LEN)           Logger answer area
XANSLEN  DC    A(ANSAA_LEN)            Answer area length
RSCODE   DS    F                       Reason code
         DSECT  ,
         IXGANSAA ,                   The answer area macro
```

Example 4

Issue IXGCONN REQUEST=DISCONNECT to disconnect from a log stream and associate some user data with the log stream.

```
          IXGCONN REQUEST=DISCONNECT,                                 X
               STREAMTOKEN=TOKEN,                                     X
               USERDATA=USERDATA,                                     X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
USERDATA DC    CL64'SOME USER DATA'   user data to log with DISCONNECT
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
TOKEN    DS    CL16                   token returned from CONNECT
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
RETCODE  DS    F                      return code from logger
RSNCODE  DS    F                      reason code from logger
```

```
DATAREA  DSECT
         IXGANSAA LIST=YES              answer area
```

Example 5

Issue IXGCONN to connect to a log stream, specifying a resource manager and resource manager exit for the log stream.

```
         L     R5,RMEXIT_ADDR
         IXGCONN REQUEST=CONNECT,
               STREAMNAME=SNAME,
               STREAMTOKEN=STOKEN,
               AUTH=WRITE,
               RMNAME=RMNAME,
               RMEXIT=(R5),
               RMDATA=RMDATA,
               RMEVENTS=(LBWRITE,LBDELETE),
               ANSAREA=XANSAREA,
               ANSLEN=XANSLEN,
               RSNCODE=RSCODE
*
SNAME    DS    CL26                    Stream name
STOKEN   DS    CL16                    Input Stream token
RMEXIT_ADDR DS A                       RM exit rtn address
RMDATA   DS    CL8                     RM exit data
XANSAREA DS    CL(ANSAA_LEN)           Logger answer area
XANSLEN  DC    A(ANSAA_LEN)            Answer area length
RSCODE   DS    F                       Reason code
         DSECT  ,
         IXGANSAA ,                   The answer area macro
```

Chapter 145. IXGDELET — Deleting log data from a log stream

Description

Use the IXGDELET macro to delete log blocks from a log stream.

For information about using the system logger services and the system logger inventory, see *z/OS MVS Programming: Assembler Services Guide*, which includes information about related macros IXGCONN, IXGBRWSE, IXGWRITE, IXGINVNT, and IXGQUERY.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key. The caller must be supervisor state with any system (0-7) PSW key to either invoke the service in SRB mode or use the MODE=SYNCEXIT keyword.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN, any SASN
AMODE:	31-bit or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameter:	All control parameters must be in the primary address space with the following exceptions: • The ECB should be addressable from the home address space. • All storage areas specified must be in the same storage key as the caller.

Programming requirements

- The current primary address space must be the same primary address space used at the time your program issued the IXGCONN request.
- The parameter list for this service must be addressable in the caller's primary address space.
- The calling program must be connected to the log stream with write authority through the IXGCONN service.
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include mapping macro IXGANSAA in your program. This macro shows the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- If there are multiple connections to a log stream, each connected application must serialize delete requests so that a delete of log blocks does not occur, for example, in the middle of another application's browse session.
- When coding the MODE=SYNCECB and ECB parameters, you must ensure that:

  - The virtual storage area specified for the ECB resides on a full word boundary.
  - You initialize the ECB field to zero.
  - The ECB resides in either the common or home address space storage at the time the IXGDELET request is issued.
  - The storage used for output parameters, such as ANSAREA and OBLOCKID, are accessible by both the IXGDELET invoker and the ECB waiter.
- When coding the MODE=SYNCEXIT parameter, you must ensure that the storage used for output parameters, such as ANSAREA and OBLOCKID, are accessible by both the IXGDELET invoker and the completion exit routine.

Restrictions

- All storage areas specified in this service must be in the same storage key as the caller's storage key and must exist in the caller's primary address space.
- There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.

Input register information

Before issuing the IXGDELET macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if register 15 contains a non-zero return code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IXGDELET macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IXGDELET.
IXGDELET	
␣	One or more blanks must follow IXGDELET.
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BLOCKS=ALL	
,BLOCKS=RANGE	
,BLOCKID=*blockid*	*blockid*: RS-type address or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,FORCE=NO	**Default**: FORCE=NO
,FORCE=YES	
,FORCEINFO=NO	**Default**: FORCEINFO=NO
,OBLOCKID=*oblockid*	*oblockid*: RS-type address or register (2) - (12).

Syntax	Description
MODE=SYNC	**Default**: MODE=SYNC
MODE=ASYNCNORESPONSE	
MODE=SYNCECB	
MODE=SYNCEXIT	
,REQDATA=*reqdata*	*reqdata*: RS-type address or register (2) - (12).
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,DIAG=NO_DIAG	**Default**: DIAG=NO_DIAG
,DIAG=NO	
,DIAG=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=0	
,PLISTVER=1	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Syntax	Description

Parameters

The parameters are explained as follows:

,STREAMTOKEN=*streamtoken*
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

,BLOCKS=ALL
,BLOCKS=RANGE
Specifies whether all or just a subset of log blocks in a log stream be deleted.

- BLOCKS=ALL: Specifies that all the log blocks in the specified log stream be deleted.
- BLOCKS=RANGE: Specifies that the range of log blocks, older than the block specified on the BLOCKID parameter, be deleted. The BLOCKID parameter is required with BLOCKS=RANGE, See *z/OS MVS Programming: Assembler Services Guide* for more information on deleting a range of log blocks.

,BLOCKID=*blockid*
Specifies the name or address (using a register) of a 8-byte input field which contains a log block identifier. BLOCKID is required with the BLOCKS=RANGE parameter. All blocks in the log stream **older** than the block specified on BLOCKID will be deleted. Note that the block specified in BLOCKID is not deleted.

Block identifiers are returned in the RETBLOCKID field of the IXGWRITE service.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,FORCE=NO
,FORCE=YES
Specifies whether this delete request can be overridden by a resource manager exit.

If you specify FORCE=NO, which is the default, the delete request **can** be overridden by the resource manager exit.

If you specify FORCE=YES, the delete request cannot be overridden by a delete exit.

,OBLOCKID=*oblockid*
Specifies the name or address (using a register) of an 8 character output field where the resource manager places the override block identifier.

,MODE=SYNC
,MODE=ASYNCNORESPONSE
,MODE=SYNCECB
,MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:

- MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- MODE=ASYNCNORESPONSE: Specifies that the request process asynchronously. The caller is not notified when the request completes and the answer area (ANSAREA) fields will not contain valid information.
- MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB parameter is posted when the request completes. The ECB parameter is required with MODE=SYNCECB.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. If the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

  The application must run in supervisor state, key 0-7 to use this parameters.

,REQDATA=*reqdata*
Specifies the name (or address in a register) of a 8-byte input field containing user-defined data to pass to the complete exit. REQDATA is only valid with the MODE=SYNCEXIT parameter.

ECB=*ecb*
Specifies the name or address (using a register) of a 4-byte input field that contains an event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB.
- The ECB must reside in either common storage or the home address space where the IXGDELET request was issued.
- The virtual storage area specified for the ECB must reside on a fullword boundary.

,DIAG=NO_DIAG
,DIAG=NO
,DIAG=YES
Specifies whether or not the DIAG option on the IXGCONN for this logstream will be in effect for this delete log data request. Refer to the DIAG keyword on the IXGINVNT, IXGCONN and IXGBRWSE macro services.

If you specify DIAG=NO_DIAG, which is the default, then the DIAG option on the IXGCONN for this logstream will be in effect for this delete log data request.

If you specify DIAG=NO, then Logger will not take additional diagnostic action as defined on the logstream definition DIAG parameter.

If you specify DIAG=YES, then Logger will take additional diagnostic action as defined on the logstream definition DIAG parameter providing the IXGCONN connect DIAG specification allows it.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, supports all parameters except those specifically referenced in higher versions.
- **2**, supports both the following parameters and parameters from version 0:
  - FORCE
  - OBLOCKID
  - REQDATA

To code: specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0 or 1

,RETCODE=*retcode*
Specifies a name or address (using a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*
Specifies a name (or address in a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When IXGDELET macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

Note: A program invoking the IXGDELET service may indicate through the MODE parameter that requests which can not be completed synchronously should have control returned to the caller prior to the completion of the request. When the request does complete, the invoker will be notified and the return and reason codes are in the answer area mapped by IXGANSAA.

The IXGCON macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

00
IXGRETCODEOK - Service completes successfully.

04
IXGRETCODEWARNING - Service completes with a warning.

08
IXGRETCODEERROR - Service does not complete.

0C
IXGRETCODECOMPERROR - Service does not complete.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 131. Return and Reason Codes for the IXGDELET Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**Equate Symbol**: IxgRsnCodeOk **Explanation**: Request processed successfully.

Table 131. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0401	**Equate Symbol**: IxgRsnCodeProcessedAsynch **Explanation**: The program specified MODE=SYNCECB, MODE=SYNCEXIT, or MODE=ASYNCNORESPONSE and the request must be processed asynchronously. **Action**: For MODE=SYNCECB, wait for the ECB specified on the ECB parameter to be posted, indicating that the request is complete. For MODE=SYNCEXIT, your complete exit is given control when the request completes. Check the ANSAA_ASYNCH_RETCODE and ANSAA_ASYNCH_RSNCODE fields, mapped by IXGANSAA, to determine whether the request completed successfully.
04	xxxx040B	**Equate Symbol**: IxgRsnCodeRMNotConnected **Explanation**: Program or environment error. The log stream is identified as being managed by a resource manager (RMNAME is specified in the active system logger couple data set policy). However, at the time of the delete request, the resource manager was not connected to the log stream and FORCE=NO was specified on the request. Delete requests can only be honored on this system if the resource manager is also connected when delete requests are being monitored. **Action**: Either: • Start the resource manager so that it can connect to the log stream. • Issue the IXGDELET request specifying FORCE=YES to delete the log block even though the resource manager is not connected to the source log stream.
04	xxxx040C	**Equate Symbol**: IxgRsnCodeRMOverrideOK **Explanation**: The caller's delete request was overridden by the associated resource manager. The override information was successfully processed.
04	xxxx040D	**Equate Symbol**: IxgRsnCodeRMNoBlock **Explanation**: Program error. The log block identifier on the IXGDELET request does not exist in the log stream. Either the block id never existed or was deleted in a previous IXGDELET request. This warning is issued only if a resource manager overrides the caller-specified block id. **Action**: Make sure that the block id specified on the IXGDELET request is correct.
04	xxxx040E	**Equate Symbol**: IxgRsnCodeRMBadGap **Explanation**: Environment error. The IXGDELET request failed because the requested log data was unreadable. This problem is caused by either an I/O error while attempting to read a DASD log data set or a log data set was deleted using an interface other than IXGDELET. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request. **Action**: System logger returns the block identifier of the first readable log block (in the direction of youngest data) in the ANSAA_GAPS_NEXT_BLKID field of the answer area mapped by IXGANSAA. If appropriate, reissue the IXGDELET request using this block identifier.

Table 131. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx040F	**Equate Symbol**: IxgRsnCodeRMEOFGap **Explanation**: Environment error. While processing the IXGDELET request, system logger prematurely reached the end or beginning of the log stream. The portion of the log stream from the requested log data to either the beginning or end of the log stream was unreadable. This problem is caused by either an I/O error while attempting to read a DASD log data set or a log data set was deleted using an interface other than IXGDELET. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request. **Action**: The action you take depends on whether your application can tolerate any loss of data. You can either: • Accept the loss of data and continue processing this log stream. • Stop using this log stream. • Correct the problem and re-issue the request.
04	xxxx0410	**Equate Symbol**: IxgRsnCodeRMLossOfDataGap **Explanation**: Environment error. The log data you tried to delete is in a section of the log stream where data is permanently missing. This condition occurs when a system or coupling facility is in recovery from a failure and not all the log data could be recovered. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. If your application can tolerate data loss, you can continue using the log stream.
04	xxxx0411	**Equate Symbol**: IxgRsnCodeRMAbended **Explanation**: Program error. The resource manager abended and percolated to the system logger recovery environment. The IXGDELET request was not processed. **Action**: Look for and correct the problem in your resource manager program or reissue the delete request, specifying FORCE=YES.
04	xxxx0412	**Equate Symbol**: IxgRsnCodeRMDisabled **Explanation**: Environment error. The log stream is identified as being managed by a resource manager (RMNAME is specified in the active system logger couple data set policy). The resource manager is connected to the log stream, but is disabled due to an abend from which it did not recover successfully (by percolating to the system logger recovery environment). **Action**: Either: • Cancel the resource manager exit and then restart the resource manager address space. • Reissue the request, specifying FORCE=YES.
04	xxxx0414	**Equate Symbol**: IxgRsnCodeRMStoppedDelete **Explanation**: The resource manager does not allow this IXGDELET request to delete any log blocks. **Action**: Determine why the resource manager is prohibiting deletes. Specify FORCE=YES to stop the resource manager exit from stopping the delete request.
08	xxxx0801	**Equate Symbol**: IxgRsnCodeBadParmlist **Explanation**: Program error. The parameter list could not be accessed. **Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller.

Table 131. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0802	**Equate Symbol**: IxgRsnCodeXESError **Explanation**: System error. A severe cross-system extended services (XES) error has occurred. **Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.
08	xxxx0804	**Equate Symbol**: IxgRsnCodeNoBlock **Explanation**: Program error. The block identifier or time stamp does not exist in the log stream. Either the value provided was never a valid location within the log stream or a prior IXGDELET request deleted the portion of the log stream it referenced. **Action**: Ensure that the value provided references an existing portion of the log stream and issue the request again. Use the LIST LOGSTREAM DETAIL(YES) request on the IXCMIPU utility to display the range of valid block identifiers for the log stream.
08	xxxx0806	**Equate Symbol**: IxgRsnCodeBadStmToken **Explanation**: Program error. One of the following occurred: • The stream token was not valid. • The specified request was issued from an address space other than the connector's address space. **Action**: Do one of the following: • Make sure that the stream token specified is valid. • Ensure the request was issued from the connector's address space.
08	xxxx080A	**Equate Symbol**: IxgRsnCodeRequestLocked **Explanation**: Program error. The program issuing the request is holding a lock. **Action**: Ensure that the program issuing the request is not holding a lock.
08	xxxx0814	**Equate Symbol**: IxgRsnCodeNotAvailForIPL **Explanation**: Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization. **Action**: See the explanation for system messages issued during system logger initialization.
08	xxxx0815	**Equate Symbol**: IxgRsnCodeNotEnabled **Explanation**: Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails. **Action**: Make sure the program issuing the request is enabled for I/O and external interrupts.
08	xxxx0816	**Equate Symbol**: IxgRsnCodeBadAnslen **Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro. **Action**: Re-issue the request, specifying an answer area of the required size.
08	xxxx0817	**Equate Symbol**: IxgRsnCodeBadAnsarea **Explanation**: Program error. The storage area specified on the ANSAREA parameter cannot be accessed. This may occur after the system logger address space has terminated. **Action**: Specify storage that is in the caller's primary address space and in the same key as the calling program at the time the system logger service was issued. This storage must be accessible until the request completes.

Table 131. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx081C	**Equate Symbol**: IxgRsnCodeNotAuthFunc **Explanation**: Program error. One of the following occurred: • The program connected to the log stream with the AUTH=READ parameter and then tried to delete or write data. You cannot write or delete data when connected with read authority. • The program connected to the log stream with the AUTH=WRITE parameter and was granted limited access (for example, write only log data and disconnect) as per the defined security access profiles, and then tried to perform a log stream access service other than IXGWRITE or IXGCONN (to disconnect). **Action**: A connection to the log stream with the appropriate access authority must be established before using the log stream delete (IXGDELET) service. If AUTH=WRITE was already in use when the error was encountered, then check with your installation's security administrator on obtaining the appropriate access to the log stream.
08	xxxx081F	**Equate Symbol**: IxgRsnCodeXcdsError **Explanation**: System error. System logger encountered an internal problem while processing the active system logger couple data set. **Action**: Contact the IBM Support Center. Provide the return and reason code and the contents of the answer area (ANSAREA field).
08	xxxx082D	**Equate Symbol**: IxgRsnCodeExpiredStmToken **Explanation**: Environment error. The stream token is no longer valid because the connector has been disconnected. **Action**: Connect to the log stream again before issuing any functional requests.
08	xxxx0836	**Equate Symbol**: IxgRsnCodeBadGap **Explanation**: Environment error. The request failed because the requested log data was unreadable. This condition could be caused by either an I/O error while attempting to read a log data set or a log data set deleted without using the IXGDELET interface. **Action**: The block identifier of the first accessible block toward the youngest data in the log stream is returned in the ANSAA_GAPS_NEXT_BLKID field in the answer area mapped by the IXGANSAA macro. If appropriate, re-issue the IXGDELET request using this block identifier.
08	xxxx083D	**Equate Symbol**: IxgRsnCodeBadECBStor **Explanation**: Program error. The ECB storage area was not accessible to the system logger. **Action**: Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller's home address space and in the same key as the caller.
08	xxxx084A	**Equate Symbol**: IxgRsnCodeEOFGap **Explanation**: Environment error. The request prematurely reached the beginning or the end of the log stream. The portion of the log stream from the requested log data to either the beginning or the end of the log stream (depending on the direction of the read) was unreadable. This condition may be caused by either an I/O error while trying to read a log data set, or a log data set deleted without using the IXGDELET interface. **Action**: The action necessary is completely up to the application depending on how critical your data is. You can do one of the following: • Accept this condition and continue reading. • Stop processing the log all together. • Attempt to get the problem rectified, if possible, and then try to re-issue the request.

Table 131. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx084B	**Equate Symbol**: IxgRsnCodeLossOfDataGap **Explanation**: Environment error. The requested log data referenced a section of the log stream where log data is permanently missing. This condition occurs when a system or coupling facility is in recovery due to a failure, but not all of the log data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
08	xxxx0861	**Equate Symbol**: IxgRsnCodeRebuildInProgress **Explanation**: Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0862	**Equate Symbol**: IxgRsnCodeXESPurge **Explanation**: Environment error. An cross-system extended services (XES) request has been purged due to re-build processing. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0863	**Equate Symbol**: IxgRsnCodeStructureFailed **Explanation**: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0864	**Equate Symbol**: IxgRsnCodeNoConnectivity **Explanation**: Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to re-build the log stream in another coupling facility or the log stream will be disconnected. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available. • The log stream has been disconnected from this system. If a re-build initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues.
08	xxxx0890	**Equate Symbol**: IxgRsnCodeAddrSpaceNotAvail **Explanation**: System error. The system logger address space failed and is not available. **Action**: Do not issue system logger requests.

Table 131. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0891	**Equate Symbol**: IxgRsnCodeAddrSpaceInitializing **Explanation**: System error. The system logger address space is not available because it is IPLing. **Action**: Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-connect to the log stream, then re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services.
08	xxxx08D0	**Equate Symbol**: IxgRsnCodeProblemState **Explanation**: Environment error. The request was rejected because of one of the following: • The request was issued in SRB mode while the requestor was in problem program state. • The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key. **Action**: Change the invoking environment to supervisor state.
08	xxxx08D1	**Equate Symbol**: IxgRsnCodeProgramKey **Explanation**: Environment error. The request was rejected because of one of the following: • The request was issued in SRB mode while the requestor was in problem program key (key 8-F). • The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key. **Action**: Change the invoking environment to a system key (key 0-7).
08	xxxx08D2	**Equate Symbol**: IxgRsnCodeNoCompleteExit **Explanation**: Program error. MODE=SYNCEXIT was specified, but the connection request did not identify a complete exit. **Action**: Either change this request to a different MODE option, or reconnect to the log stream with a complete exit specified on the COMPLETEXIT parameter.
08	xxxx085F	**Equate Symbol**: IxgRsnPercToRequestor **Explanation**: Environment error. Percolation to the service requestor's task occurred because of an abend during system logger processing. Retry was not allowed. **Action**: Issue the request again. If the problem persists, contact the IBM Support Center.
0C	xxxx0000	**Equate Symbol**: IxgRetCodeCompError **Explanation**: User or System error. One of the following occurred: • You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space. • System logger component error occurred. **Action**: If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSAA) and any dumps or LOGREC entries from system logger.

Examples

Example 1

Delete all data from the log stream.

```
          IXGDELET STREAMTOKEN=TOKEN,                                  X
               BLOCKS=ALL,                                             X
               MODE=SYNC,                                              X
               ANSAREA=ANSAREA,                                        X
               ANSLEN=ANSLEN,                                          X
               RSNCODE=RSNCODE,                                        X
               MF=S,                                                   X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)          length of logger's answer area
TOKEN    DS    CL16                  stream token from connect
ANSAREA  DS    CL(ANSAA_LEN)         answer area for log requests
RETCODE  DS    F                     return code
RSNCODE  DS    F                     reason code
DATAREA  DSECT
         IXGANSAA LIST=YES           answer area
```

Example 2

Delete a range of data from the log stream asynchronously, if synchronous processing is not possible.

```
          IXGDELET STREAMTOKEN=TOKEN,                                  X
               BLOCKS=RANGE,                                           X
               BLOCKID=BLOCKID,                                        X
               MODE=SYNCECB,                                           X
               ECB=ANECB,                                              X
               ANSAREA=ANSAREA,                                        X
               ANSLEN=ANSLEN,                                          X
               RSNCODE=RSNCODE,                                        X
               MF=S,                                                   X
               RETCODE=RETCODE
*****************************************************************
*         If rsncode = '00000401'X then wait on
*         the ecb ANECB.
*****************************************************************
ANSLEN   DC    A(L'ANSAREA)          length of logger's answer area
BLOCKID  DS    CL8                   block id from which to delete
TOKEN    DS    CL16                  stream token from connect
ANSAREA  DS    CL(ANSAA_LEN)         answer area for log requests
ANECB    DS    F                     ecb on which to wait
RETCODE  DS    F                     return code
RSNCODE  DS    F                     reason code
DATAREA  DSECT
         IXGANSAA LIST=YES           answer area
```

Example 3

Delete all data from the log stream using registers with the macro.

```
          LA   R6,TOKEN              load stream token into register 6
          IXGDELET STREAMTOKEN=(6),                                    X
               BLOCKS=ALL,                                             X
               MODE=SYNC,                                              X
               ANSAREA=ANSAREA,                                        X
               ANSLEN=ANSLEN,                                          X
               RSNCODE=RSNCODE,                                        X
               MF=S,                                                   X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)          length of logger's answer area
TOKEN    DS    CL16                  stream token from connect
ANSAREA  DS    CL(ANSAA_LEN)         answer area for log requests
RETCODE  DS    F                     return code
RSNCODE  DS    F                     reason code
DATAREA  DSECT
         IXGANSAA LIST=YES           answer area
R6       EQU   6
```

Chapter 146. IXGWRITE — Write log data to a log stream

Description

Use the IXGWRITE macro to allow a program to write a log block to a log stream. IXGWRITE returns a unique identifier for each log block written to the log stream.

System logger generates a time stamp for each log block as they are received from applications issuing IXGWRITE and writes the blocks to the log stream in that order. Applications that imbed their own time stamps in log blocks will find that the blocks may not be in application-generated time stamp order, especially if multiple applications are writing to a log stream simultaneously. In order to ensure chronological order of log blocks by application-generated time stamp, applications should provide their own serialization on the log stream.

For information on using the system logger services and the LOGR policy, see *z/OS MVS Programming: Assembler Services Guide*, which also includes information about related macros IXGCONN, IXGBRWSE, IXGINVNT, IXGDELET, and IXGQUERY.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key. The caller must be supervisor state with any system (0-7) PSW key to either invoke this service in SRB mode or to use the MODE=SYNCEXIT keyword.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.
Control parameters:	All control parameters must be in the primary address space with the following exceptions: • The ECB should be addressable from the home address space. • Any parameter area that is explicitly ALET-qualified as allowed by the input parameter (for example, the area referenced by the BUFFER parameter when the BUFFALET parameter is specified) must be in an address or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All storage areas specified must be in the same storage key as the caller, with the following exception: The parameter area is explicitly storage key qualified as allowed by the input parameters (example: the area referenced by the BUFFER parameter when the BUFFKEY parameter is also specified).

Programming requirements

- Before issuing IXGWRITE, you must put the data you wish to write to the log stream into a buffer specified on the BUFFER parameter. IXGWRITE will then write this buffer to the log stream as a log block.
- The current primary address space from which you issue the IXGWRITE service must be the same as the primary address space at the time you issued the IXGCONN request.
- The parameter list for this service must be addressable in the caller's primary address space.
- The calling program must be connected to the log stream with write authority through the IXGCONN service.
- IXGWRITE cannot be issued if the connection is an import connection (IMPORTCONNECT=YES on the IXGCONN service). The IXGWRITE service must be issued under a write connection (IMPORTCONNECT=NO, which is the default).
- Include the IXGCON mapping macro in your program. This macro provides a list of equate symbols for the system logger services.
- Include mapping macro IXGANSAA in your program. This macro shows the format of the answer area output returned for each system logger service in the ANSAREA parameter.
- When coding the MODE=SYNCECB and ECB parameters, you must ensure that:
  - The virtual storage area specified for the ECB resides on a full word boundary.
  - You initialize the ECB field to zero.
  - The ECB resides in either the common or home address space storage at the time the IXGWRITE request is issued.
  - The storage used for output parameters, such as ANSAREA, RETBLOCKID, and TIMESTAMP, are accessible by both the IXGWRITE invoker and the ECB waiter.
- When coding the MODE=SYNCEXIT parameter, you must ensure that the storage used for output parameters, such as ANSAREA, RETBLOCKID, and TIMESTAMP, are accessible by both the IXGWRITE invoker and the completion exit routine.

Restrictions

- All storage areas specified on this macro must be in the same storage key as the caller's storage key, with the exception of the BUFFKEY parameter.

  Storage areas that are not ALET-qualified must exist in the caller's primary address space. The ECB should be addressable from the home address space.
- There is more than one version of this macro available. The parameters you can use depend on the version you specify on the PLISTVER parameter. See the description of the PLISTVER parameter for more information.
- You can call any of the system logger services in either AMODE 31 or 64, but the parameter list and all other data addresses, with the excption of BUFFER64 must reside in 31-bit storage.

Input register information

Before issuing the IXGWRITE macro, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if register 15 contains a non-zero return code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as a work register by the system

2-13
Unchanged

14-15
Used as a work register by the system

Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

The standard form of the IXGWRITE macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede IXGWRITE.
IXGWRITE	
␣	One or more blanks must follow IXGWRITE.
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BUFFER=*buffer*	*buffer*: RS-type address or register (2) - (12).
BUFFER64=*buffer64*	*buffer64*: RS-type address or register (2) - (12).
,BLOCKLEN=*blocklen*	*blocklen*: RS-type address or register (2) - (12).

Syntax	Description
,RETBLOCKID=*retblockid*	*retblockid*: RS-type address or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,BUFFKEY=*buffkey*	*buffkey*: RS-type address or register (2) - (12).
,BUFFKEY=*	**Default**: BUFFKEY=*
,BUFFALET=*buffalet*	*buffalet*: RS-type address or register (2) - (12). **Default**: BUFFALET=0
,TIMESTAMP=*timestamp*	*timestamp*: RS-type address or register (2) - (12). **Default**: NO_TIMESTAMP
MODE=SYNC	**Default**: MODE=SYNC
MODE=ASYNCNORESPONSE	
MODE=SYNCECB	
MODE=SYNCEXIT	
,REQDATA=*reqdata*	*reqdata*: RS-type address or register (2) - (12).
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	
,PLISTVER=0	
,PLISTVER=1	
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	**Default**: MF=S

Syntax	Description
,MF=(L,*list addr*)	
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained as follows:

,STREAMTOKEN=*streamtoken*
Specifies the name (or address in a register) of a required 16-byte input field containing the token for the log stream that you want to write to. The stream token is returned by the IXGCONN service at connection to the log stream.

,BUFFER=*buffer*
,BUFFER64=*buffer64*
Specifies the field name (or address in a register) of the data to be written to the log.

- BUFFER=*buffer* specifies that the location of the buffer is in 31-bit storage.
- BUFFER64=*buffer64* specifies that the location of the buffer is in 64-bit storage.

The BUFFER and BUFFER64 parameters are mutually exclusive.

,BLOCKLEN=*blocklen*
Specifies the name (or address in a register) of a 4-byte input field that contains the length in bytes of the log block you are writing to the log stream.

The value of BLOCKLEN must be between 1 and the value for MAXBUFSIZE.

RETBLOCKID=*retblockid*
Specifies the name (or address in a register) of a 8-byte output field where IXGWRITE returns the unique block identifier for the log block written to the log stream.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

BUFFALET=*buffalet*
Specifies the name (or address in a register) of a 4-byte input field specifying the access list entry table (ALET) to be used to access the buffer specified on the BUFFER or BUFFER64 keyword. If the buffer is ALET-qualified, the ALET must index a valid entry on the task's dispatchable unit access list

(DUAL) or specify a SCOPE=COMMON data space. An ALET that indexes the system logger PASN-AL list will not work.

The default is 0, which means that the buffer is in the calling program's primary address space.

BUFFKEY=*buffkey*
Specifies the name (or address in a register) of a 4-byte input field specifying the storage key for the buffer specified on the BUFFER or BUFFER64 parameter.

If the caller is running in problem state, the caller's PSW key and the key specified in the BUFFKEY parameter must match.

If the caller is running in supervisor state, specify any syntactically valid (0 through 15) key on the BUFFKEY parameter.

If you omit the BUFFKEY parameter, the default used is the PSW key of the caller.

TIMESTAMP=*timestamp*
Specifies the name (or address in a register) of a 16-byte output field where the Greenwich mean time and local time stamps associated with the requested log block are returned when the write request is successful. Both time stamps will be in time of day (TOD) clock format.

MODE=SYNC
MODE=ASYNCNORESPONSE
MODE=SYNCECB
MODE=SYNCEXIT
Specifies that the request should be processed in one of the following ways:

- MODE=SYNC: Specifies that the request process synchronously. Control is not returned to the caller until request processing is complete. If necessary, the calling program will be suspended until the request completes.
- MODE=ASYNCNORESPONSE: Specifies that the request process asynchronously. The caller is not notified when the request completes and the answer area (ANSAREA) fields will not contain valid information.
- MODE=SYNCECB: Specifies that the request process synchronously if possible. If the request processes asynchronously, control returns to the caller before the request completes and the event control block (ECB) specified on the ECB keyword is posted when the request completes. The ECB keyword is required with MODE=SYNCECB.
- MODE=SYNCEXIT: Specifies that the request process synchronously, if possible. If the request cannot be processed synchronously, your complete exit (specified on the COMPLETEEXIT parameter on the IXGCONN request) gets control when this request completes. Control returns to the caller with a return and reason code indicating that the request is not complete. The system passes the data specified on the REQDATA parameter, if specified, to the complete exit.

  When a MODE=SYNCEXIT request processes asynchronously, system logger maintains latent binds to the storage location specified by the answer area (ANSAREA) fields, and, if specified, to RETBLOCKID and TIMESTAMP.

  The application must run in supervisor state, key 0-7 to use this parameter.

,REQDATA=*reqdata*
Specifies the name (or address in a register) of a 8-byte input field containing user-defined data to pass to the complete exit. REQDATA is only valid with the MODE=SYNCEXIT parameter.

,ECB=*ecb*
Specifies the name (or address in a register) of a 4-byte input field that contains the event control block (ECB) to be posted when the request completes.

Before coding ECB, you must ensure that:

- You initialize the ECB to zero.
- The ECB must reside in either common storage or the home address space where the IXGWRITE service was issued.

- The virtual storage area specified for the ECB must reside on a fullword boundary.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1

An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates.

The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Note that on the list form, the default will cause the smallest parameter list to be created.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form when both forms are assembled using the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, which supports all parameters except those specifically referenced in higher versions.
- **1**, which supports both the following parameters and parameters from version 0:
  - REQDATA

To code: Specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0 or 1

,RETCODE=*retcode*

Specifies a name (or address in a register) of a 4-byte output field where the system will place the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsncode*

Specifies a name (or address in a register) of a 4-byte output field where the system will place the reason code. The reason code is also in general purpose register (GPR) 0, if you received a non-zero return code.

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can be specified on the list form of the macro. IBM recommends that you always specify PLISTVER=MAX on the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

IBM recommends that you use the modify and execute forms in the following order:

- Use MF=(M,*list_addr*,COMPLETE), specifying appropriate parameters, including all required ones.
- Use MF=(M,*list_addr*,NOCHECK), specifying the parameters you want to change.
- Use MF=(E,*list_addr*,NOCHECK), to execute the macro.

,list addr
The name of a storage area to contain the parameters.

,attr
An optional 1- to 60-character input string, which can contain any value that is valid on an assembler DS pseudo-op. You can use this parameter to force boundary alignment of the parameter list. If you do not code *attr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

,COMPLETE
Specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
Specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

ABEND codes

None.

Return and reason codes

When IXGWRITE macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

Note: A program invoking the IXGWRITE service may indicate through the MODE parameter that requests which can not be completed synchronously should have control returned to the caller prior to completion of the request. When the request does complete, the invoker will be notified and the return and reason codes are in the answer area mapped by IXGANSAA.

The IXGCON macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

00
IXGRSNCODEOK - Successful completion.

04
IXGRSNCODEWARNING - The request was processed successfully, however a warning condition was encountered.

08
IXGRETCODEERROR - An error has been encountered. The associated reason code provides more information.

0C
IXGRETCODECOMPERROR - A system logger component error has been encountered.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 132. Return and reason codes for the IXGWRITE macro

Return code	Reason code	Meaning and action
00	xxxx0000	**Equate symbol**: IxgRsnCodeOk **Explanation**: Request processed successfully.
04	xxxx0401	**Equate symbol**: IxgRsnCodeProcessedAsynch **Explanation**: The program specified MODE=SYNCECB, MODE=SYNCEXIT, or MODE=ASYNCNORESPONSE and the request must be processed asynchronously. **Action**: For MODE=SYNCECB, wait for the ECB specified on the ECB parameter to be posted, indicating that the request is complete. For MODE=SYNCEXIT, your complete exit is given control when the request completes. Check the ANSAA_ASYNCH_RETCODE and ANSAA_ASYNCH_RSNCODE fields, mapped by IXGANSAA, to determine whether the request completed successfully.
04	xxxx0405	**Equate symbol**: IxgRsnCodeWarningLossOfData **Explanation**: Environment error. Returned for READCURSOR, START OLDEST and RESET OLDEST requests. This condition occurs when a system and coupling facility fail and not all of the log data in the log stream could be recovered. • For READCURSOR: A log block has been returned, but there may be log blocks permanently missing between this log block and the one previously returned. • For START OLDEST and RESET OLDEST: The oldest log blocks in the log stream may be permanently missing, the browse cursor is set at the oldest available log block. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
04	xxxx0407	**Equate symbol**: IxgRsnCodeConnPossibleLossOfData **Explanation**: Environment error. The request was successful, but there may be log blocks permanently missing between this log block and the one previously returned. This condition occurs when a system or coupling facility fails and not all of the data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
04	xxxx0408	**Equate symbol**: IxgRsnCodeDsDirectoryFullWarning **Explanation**: Environment error. The request was successful, but the log streams DASD data set directory is full. System logger cannot offload any further data from the coupling facility structure to DASD. The system logger will continue to process IXGWRITE requests until this log streams portion of the coupling facility structure becomes full. **Action**: Either delete enough data from the log stream to free up space in the log streams data set directory so that offloading can occur or disconnect from the log stream.
04	xxxx0409	**Equate symbol**: IxgRsnCodeWowWarning **Explanation**: Environment error. The request was successful, but an error condition was detected during a previous offload of data. System logger might not be able to offload further data. System logger will continue to process IXGWRITE requests only until the interim storage for the log stream is filled. (Interim storage is the coupling facility for a coupling facility log stream and local storage buffers for a DASD-only log stream.) **Action**: Do not issue any further requests for this log stream and disconnect. Connect to another log stream. Check the system log for message IXG301I to determine the cause of the error. If you cannot fix the error, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.

Table 132. Return and reason codes for the IXGWRITE macro (continued)

Return code	Reason code	Meaning and action
04	0000040A	**Equate symbol**: IxgRsnCodeDuplexFailureWarning **Explanation**: Environment error. The request was successful, but the system logger was unable to duplex log data to staging data sets, even though the log stream definition requested unconditional duplexing to staging data sets by specifying the log stream attributes: STG_DUPLEX=YES, DUPLEXMODE=UNCOND, or STG_DUPLEX=YES,DUPLEXMODE=DRXRC. When DUPLEXMODE=UNCOND is specified, but Logger was unable to obtain a staging data set to duplex the log data. Therefore, the Logger duplexing is being done in local buffers (data space). When DUPLEXMODE=DRXRC is specified for a logstream and being used for (non-local) disaster recovery duplexing, if the internal buffers used for asynchronous buffering of the log blocks become full. Meaning the internal buffers became full before at least one of the full buffers could be written to the staging data set. **Action**: For DUPLEXMODE=UNCOND, if duplexing to staging data sets is required, disconnect from this log stream and connect to a log stream that can be duplexed to staging data sets. For DUPLEXMODE=DRXRC, if duplexing to a DRXRC-type staging data sets is required, then cause the log data to be offload to the log stream secondary storage (offload data sets) and then continue writing to the log stream.
08	xxxx0801	**Equate symbol**: IxgRsnCodeBadParmlist **Explanation**: Program error. The parameter list could not be accessed. **Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx0802	**Equate symbol**: IxgRsnCodeXESError **Explanation**: System error. A severe cross-system extended services (XES) error has occurred. **Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.
08	xxxx0803	**Equate symbol**: IxgRsnCodeBadBuffer **Explanation**: Program error. The virtual storage area specified on the BUFFER or BUFFER64 parameter is not addressable. **Action**: Ensure that the storage area specified on the BUFFER or BUFFER64 parameter is accessible to system logger for the duration of the request. If the BUFFKEY parameter is specified, make sure it contains a valid key associated with the storage area. If BUFFKEY is not used, ensure that the storage is in the same key as the program at the time the logger service was requested. The storage must be addressable in the caller's primary address space.
08	xxxx0806	**Equate symbol**: IxgRsnCodeBadStmToken **Explanation**: Program error. One of the following occurred: • The stream token was not valid. • The specified request was issued from an address space other than the connector's address space. **Action**: Do one of the following: • Make sure that the stream token specified is valid. • Ensure the request was issued from the connector's address space.

Table 132. Return and reason codes for the IXGWRITE macro (continued)

Return code	Reason code	Meaning and action
08	xxxx0809	**Equate symbol**: IxgRsnCodeBadWriteSize **Explanation**: Program error. The size of the log block specified in the BLOCKLEN parameter is not valid. The value for BLOCKLEN must be greater than zero and less than or equal to the maximum buffer size (MAXBUFSIZE) defined in the active system logger couple data set policy for the log stream or for the structure associated with this log stream. **Action**: Ensure that the value specified on the BLOCKLEN parameter is greater than 0 and less than or equal to the MAXBUFSIZE which is returned on the log stream connect request.
08	xxxx080A	**Equate symbol**: IxgRsnCodeRequestLocked **Explanation**: Program error. The program issuing the request is holding a lock. **Action**: Ensure that the program issuing the request is not holding a lock.
08	xxxx0814	**Equate symbol**: IxgRsnCodeNotAvailForIPL **Explanation**: Environment error. The system logger address space is not available for the remainder of this IPL. The system issues messages about this error during system logger initialization. **Action**: See the explanation for system messages issued during system logger initialization.
08	xxxx0815	**Equate symbol**: IxgRsnCodeNotEnabled **Explanation**: Program error. The program issuing the request is not enabled for I/O and external interrupts, so the request fails. **Action**: Make sure the program issuing the request is enabled for I/O and external interrupts.
08	xxxx0816	**Equate symbol**: IxgRsnCodeBadAnslen **Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro. **Action**: Re-issue the request, specifying an answer area of the required size.
08	xxxx0817	**Equate symbol**: IxgRsnCodeBadAnsarea **Explanation**: Program error. The storage area specified on the ANSAREA parameter cannot be accessed. This may occur after the system logger address space has terminated. **Action**: Specify storage that is in the caller's primary address space and in the same key as the calling program at the time the system logger service was issued. This storage must be accessible until the request completes.
08	xxxx0818	**Equate symbol**: IxgRsnCodeBadBlockidStor **Explanation**: Program error. The storage area specified by BLOCKID cannot be accessed. **Action**: Ensure that the storage area is accessible to system logger for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx081C	**Equate symbol**: IxgRsnCodeNotAuthFunc **Explanation**: Program error. The program connected to the log stream with the AUTH=READ parameter and then tried to delete or write data. You cannot write or delete data when connected with read authority. **Action**: Issue the IXGCONN service with AUTH=WRITE authority and then re-issue this request.

Table 132. Return and reason codes for the IXGWRITE macro (continued)

Return code	Reason code	Meaning and action
08	xxxx082D	**Equate symbol**: IxgRsnCodeExpiredStmToken **Explanation**: Environment error. The stream token is no longer valid because the connector has been disconnected. **Action**: Connect to the log stream again before issuing any functional requests.
08	xxxx0837	**Equate symbol**: IxgRsnCodeBadTimestamp **Explanation**: Program error. The storage area specified by TIMESTAMP cannot be accessed. **Action**: Ensure that the storage area is accessible to the system logger service for the duration of the request. The storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx083D	**Equate symbol**: IxgRsnCodeBadECBStor **Explanation**: Program error. The ECB storage area was not accessible to the system logger. **Action**: Ensure that the storage area is accessible to the system logger for the duration of the request. The storage must be addressable in the caller's home address space and in the same key as the caller.
08	xxxx083F	**Equate symbol**: IxgRsnCodeTestartError **Explanation**: System error. An unexpected error was encountered while attempting to validate the buffer ALET. **Action**: See ANSAA_DIAG1 in the answer area mapped by the IXGANSAA macro for the return code from the TESTART system service.
08	xxxx0841	**Equate symbol**: IxgRsnCodeBadBufferAlet **Explanation**: Program error. The buffer ALET specified is not zero and does not represent a valid entry on the caller's dispatchable unit access list (DUAL). See the ANSAA_DIAG1 field of the answer area, mapped by the IXGANSAA macro, for the return code from the TESTART system service. **Action**: Ensure that the correct ALET was specified. If not, provide the correct ALET. Otherwise, add the correct ALET to dispatchable unit access list (DUAL).
08	xxxx0849	**Equate symbol**: IxgRsnCodeBadBuffkey **Explanation**: Program error. The buffer key specified on the BUFFKEY parameter specifies an invalid key. Either the key is greater than 15 or the program is running in problem state and the specified key is not the same key as the PSW key at the time the system logger service was issued. **Action**: For problem state programs, either do not specify the BUFFKEY parameter or else specify the same key as the PSW key at the time the system logger service was issued. For supervisor state programs, specify a valid storage key (0 <= key <= 15).
08	xxxx084E	**Equate symbol:** IXGRSNCODESTRSACETOOSMALL **Explanation:** Environment error. Structure resources are not available to satisfy the request. All structure resources are allocated as system logger control resources. This condition occurs when the structure resources are consumed by the logstreams connections. **Action:** Increase the size of the structure in the CFRM policy or use the SETXCF ALTER command to dynamically increase the size of the structure.

Table 132. Return and reason codes for the IXGWRITE macro (continued)

Return code	Reason code	Meaning and action
08	xxxx085C	**Equate symbol**: IxgRsnCodeDsDirectoryFull **Explanation**: Environment error. The interim storage (for example: the coupling facility structure space allocated or the staging data set space) for the log stream is full. System logger's attempts to offload the interim storage log data to DASD has failed because the log stream's data set directory is full. If this reason code is issued by the IXGWRITE request, no further write requests can be processed until additional directory space is available for the log stream. System logger will periodically re-drive its offload attempts for this condition, which is applicable to both coupling facility structure and DASD-only type log streams. If system logger is able to offload log data, then an ENF event will be issued informing the connectors that the log stream should be available for writing more log data. However, the time that passes before you can write to the log stream is unpredictable. The system issues related messages IXG257I, IXG261E, IXG262A and IXG301I. **Action**: The system programmer must make more log stream data set directory space available. For information about how an authorized application program might respond to this reason code, see Setting up the system logger configuration in the *z/OS MVS Programming: Authorized Assembler Services Guide*. For information about how an unauthorized application program might respond to this reason code, see IXGWRITE: Writing to a log stream in the *z/OS MVS Programming: Assembler Services Guide*.
08	xxxx085D	**Equate symbol**: IxgRsnCodeWowError **Explanation**: Environment error. The interim storage (for example: the coupling facility structure space allocated or the staging data set space) for the log stream is full. System logger's attempts to offload the interim storage log data to DASD have failed because of severe errors. No further write requests can be processed until the offload error condition is cleared. System logger will periodically re-drive its offload attempts for this condition, which is applicable to both coupling facility structure and DASD-only type log streams. If system logger is able to offload log data, then an ENF event will be issued informing the connectors that the log stream should be available for writing more log data. However, the time that passes before you can write to the log stream is unpredictable. The system issues related message IXG301I. **Action**: The system programmer must correct the severe error condition inhibiting the log stream offload. If you are unable to correct the error, search problem reporting data bases for a fix for the problem. If no fixt exists, contact the IBM Support Center. You can retry your write request periodically or wait for the ENF signal that the log stream is available, or disconnect from this log stream and connect to another log stream. For information on how an authorized application program might respond to this reason code, see Setting up the system logger configuration in the *z/OS MVS Programming: Authorized Assembler Services Guide*. For information on how an authorized application program might respond to this reason code, see IXGWRITE: Writing to a log stream in the *z/OS MVS Programming: Assembler Services Guide*.

Table 132. Return and reason codes for the IXGWRITE macro (continued)

Return code	Reason code	Meaning and action
08	xxxx0860	**Equate symbol**: IxgRsnCodeCFLogStreamStorFull **Explanation**: Environment error. The coupling facility structure space allocated for this log stream is full. No further requests can be processed until the log data in the coupling facility structure is offloaded to DASD log data sets. **Action**: Listen to the ENF signal 48 which will indicate that the log stream is available after the data has been offloaded to DASD. For IXGCONN requests, Listen to the ENF signal 48 which will indicate that the structure is available. Then, re-issue the request.
08	xxxx0861	**Equate symbol**: IxgRsnCodeRebuildInProgress **Explanation**: Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0862	**Equate symbol**: IxgRsnCodeXESPurge **Explanation**: Environment error. An cross-system extended services (XES) request has been purged due to re-build processing. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0863	**Equate symbol**: IxgRsnCodeStructureFailed **Explanation**: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0864	**Equate symbol**: IxgRsnCodeNoConnectivity **Explanation**: Environment error. No connectivity exists to the coupling facility associated with the log stream. The system logger will either attempt to re-build the log stream in another coupling facility or the log stream will be disconnected. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available. • The log stream has been disconnected from this system. If a re-build initiated because of a loss of connectivity previously failed, an ENF corresponding to this reason code might not be issued. Further action by the installation might be necessary to cause the change of the log stream status again. Check the log for messages IXG101I, IXG107I and related rebuild messages for information on resolving any outstanding issues.

Table 132. Return and reason codes for the IXGWRITE macro (continued)

Return code	Reason code	Meaning and action
08	xxxx0865	**Equate symbol**: IxgRsnCodeStagingDSFull **Explanation**: Environment error. The staging data set allocated for this log stream on this system is full. No further requests can be processed until enough log data in the coupling facility structure is offloaded to DASD log data sets to relieve the staging data set's full condition. **Action**: Listen to the ENF signal 48 which will indicate that the log stream is available after room becomes available in the staging data set. Then, re-issue the request.
08	xxxx0867	**Equate symbol**: IxgRsnCodeLocalBufferFull **Explanation**: Environment error. One of the two following problems was detected: • The available local buffer space (data space storage) for the system logger address space is full. Ansaa_Diag1 and Ansaa_Diag2 in the Answer Area will contain 0 for this error return. • The IXGWRITE is rejected because a caller attempted to write log data when the outstanding asynchronous write activity for this connection was considered too high. The limit for unauthorized IXGWRITE invokers is 2,000 and the limit of 10,000 is used for authorized callers. An unauthorized caller is a caller whose PSW key is >= 8 and that is not in supervisor state. ANSAA_DIAG1 in the answer area contains a value of 1 for this error return for unauthorized callers and a value of 2 for authorized callers. ANSAA_DIAG2 contains the total number of outstanding write requests for this connection. No further writing requests can be processed until the log data in the local buffer space is offloaded to DASD log data sets or this connector's prior IXGWRITE requests complete. **Note:** This reason code applies to both CF and DASD only log stream requests. **Action**: • For authorized writers: Listen for the ENF signal 48 that will indicate that the log stream is available. With the first condition, logger issues the ENF signal after the data has been offloaded to DASD. With the second condition, logger issues the ENF signal 48 that the log stream is available when the number of in-flight authorized asynchronous writes is reduced below 85% of the limit. There will be no ENF signal issued when the unauthorized limit is relieved. • For unauthorized callers: Wait for a short interval and then reissue the request. • If the attempts continue to fail or the ENF signal is not issued for an unacceptable period, consider notifying operations or disconnecting from the log stream.
08	xxxx0868	**Equate symbol**: IxgRsnCodeStagingDSFormat **Explanation**: Environment error. The staging data set allocated for this log stream on this system has not finished being formatted for use by System Logger. No further IXGWRITE requests can be processed until the formatting completes. **Action**: Listen to the ENF signal 48 which will indicate that the logstream is available after formatting process is finished. Then, re-issue the request.
08	xxxx0890	**Equate symbol**: IxgRsnCodeAddrSpaceNotAvail **Explanation**: System error. The system logger address space failed and is not available. **Action**: Do not issue system logger requests.

Table 132. Return and reason codes for the IXGWRITE macro (continued)

Return code	Reason code	Meaning and action
08	xxxx0891	**Equate symbol**: IxgRsnCodeAddrSpaceInitializing **Explanation**: System error. The system logger address space is not available because it is IPLing. **Action**: Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-connect to the log stream, then re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services.
08	xxxx08D1	**Equate symbol**: IxgRsnCodePrgramKey **Explanation**: Environment error. The request was rejected because of one of the following: • The request was issued in SRB mode while the requestor was in problem program key (key 8-F). • The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key. **Action**: Change the invoking environment to a system key (key 0-7).
08	xxxx08D2	**Equate symbol**: IxgRsnCodeNoCompleteExit **Explanation**: Program error. MODE=SYNCEXIT was specified, but the connection request did not identify a complete exit. **Action**: Either change this request to a different MODE option, or reconnect to the log stream with a complete exit specified on the COMPLETEXIT parameter.
08	xxxx08D7	**Equate symbol**: IxgRsnCodeRequestNotAllowed **Explanation**: Program error. The caller issued an IXGWRITE request while an import connection was active on this system (IXGCONN IMPORTCONNECT=YES). **Action**: Re-issue the request, based on the type of connection active.
0C	xxxx0000	**Equate symbol**: IxgRetCodeCompError **Explanation**: User or System error. One of the following occurred: • You issued the FORCE IXGLOGR,ARM command to terminate the system logger address space. • System logger component error occurred. **Action**: If this reason code is not the result of forcing the system logger address space, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center. Provide the diagnostic data in the answer area (IXGANSAA) and any dumps or LOGREC entries from system logger.

Example 1

Write data to the log stream synchronously.

```
          IXGWRITE STREAMTOKEN=TOKEN,                                  X
               BUFFER=BUFF,                                            X
               BLOCKLEN=BLKLEN,                                        X
               BUFFALET=BUFALET,                                       X
               RETBLOCKID=RETBLK,                                      X
               BUFFKEY=BUFKEY,                                         X
               TIMESTAMP=RET_TIME,                                     X
               MODE=SYNC,                                              X
               ANSAREA=ANSAREA,                                        X
               ANSLEN=ANSLEN,                                          X
               RSNCODE=RSNCODE,                                        X
               MF=S,                                                   X
               RETCODE=RETCODE
BUFF     DC    CL256'BUFFER TEXT'     buffer to write to log stream
BLKLEN   DC    F'256'                 length of block to be written
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
```

```
BUFKEY   DC    F'8'                  buffer key
TOKEN    DS    CL16                  stream token from connect
RET_TIME DS    CL16                  returned timestamp of block
ANSAREA  DS    CL(ANSAA_LEN)         answer area for log requests
RETCODE  DS    F                     return code
RSNCODE  DS    F                     reason code
BUFALET  DC    F'1'                  buffer alet secondary
RETBLK   DS    CL8                   returned block id
DATAREA  DSECT
         IXGANSAA LIST=YES           answer area
```

Example 2

Write data to the log stream asynchronously, if synchronous processing is not possible.

```
          IXGWRITE STREAMTOKEN=TOKEN,                                  X
               BUFFER=BUFF,                                            X
               BLOCKLEN=BLKLEN,                                        X
               BUFFALET=BUFALET,                                       X
               RETBLOCKID=RETBLK,                                      X
               MODE=SYNCECB,                                           X
               ECB=ANECB,                                              X
               ANSAREA=ANSAREA,                                        X
               ANSLEN=ANSLEN,                                          X
               RSNCODE=RSNCODE,                                        X
               MF=S,                                                   X
               RETCODE=RETCODE
*****************************************************************
*          if return code = '00000401'X then wait
*          on the ecb ANECB for the request to complete
*****************************************************************
BUFF     DC    CL256'BUFFER TEXT'    buffer to write to log stream
BLKLEN   DC    F'256'                length of block to be written
ANSLEN   DC    A(L'ANSAREA)          length of logger's answer area
TOKEN    DS    CL16                  stream token from connect
ANSAREA  DS    CL(ANSAA_LEN)         answer area for log requests
RETCODE  DS    F                     return code
RSNCODE  DS    F                     reason code
BUFALET  DC    F'1'                  buffer alet secondary
ANECB    DS    F                     ecb to wait on
RETBLK   DS    CL8                   returned block id
DATAREA  DSECT
         IXGANSAA LIST=YES           answer area
```

Example 3

Write data to the log stream using registers.

```
          LA R6,TOKEN                load stream token in register 6
          IXGWRITE STREAMTOKEN=(6),                                    X
               BUFFER=BUFF,                                            X
               BLOCKLEN=BLKLEN,                                        X
               RETBLOCKID=RETBLK,                                      X
               MODE=SYNC,                                              X
               ANSAREA=ANSAREA,                                        X
               ANSLEN=ANSLEN,                                          X
               RSNCODE=RSNCODE,                                        X
               MF=S,                                                   X
               RETCODE=RETCODE
BUFF     DC    CL256'BUFFER TEXT'    buffer to write to log stream
BLKLEN   DC    F'256'                length of block to be written
ANSLEN   DC    A(L'ANSAREA)          length of logger's answer area
TOKEN    DS    CL16                  stream token from connect
ANSAREA  DS    CL(ANSAA_LEN)         answer area for log requests
RETCODE  DS    F                     return code
RSNCODE  DS    F                     reason code
RETBLK   DS    CL8                   returned block id
DATAREA  DSECT
         IXGANSAA LIST=YES           answer area
R6       EQU   6                     set up register 6
```

Appendix A. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Documentation with 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 are considered 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 the 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`, 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 is placed 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 to provide 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, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line `2.1 %OP1` means that you must refer to separate syntax fragment `OP1`.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark 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 the 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.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the `!` symbol and a syntax element indicate 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 dotted decimal number can specify the ! 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 the example, if you include the `FILE` keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the `FILE` keyword is omitted, the 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 applies only 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.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero 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, 3 STATE`, you know that you can include `HOST, STATE`, both together, or nothing.

Notes:

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 can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the 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 repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA 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 grant you any license to these patents. You can send license inquiries, in writing, to:

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

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

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

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.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for IBM Documentation. Use of hyperlinks in other output formats of this information is at your own risk.

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
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 document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

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.

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

COPYRIGHT LICENSE:

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

Terms and conditions for product documentation

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

Applicability

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

Personal use

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

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

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

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

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

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

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, 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 (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

This information is intended to help the customer to code macros that are available to authorized assembler language programs. This information documents intended programming interfaces that allow the customer to write programs to obtain 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 (www.ibm.com/legal/copytrade.shtml).

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

Index

Numerics

A

C

D

E

F

G

H

I

J

K

L

M

N

S

T

U

V

W

X

Z

Product Number: 5650-ZOS

SA23-1373-40