z/OS

MVS Programming: Authorized Assembler Services Reference, Volume 2 (ENFREQ-IXGWRITE)

SA22-7610-03

z/OS

MVS Programming: Authorized Assembler Services Reference, Volume 2 (ENFREQ-IXGWRITE)

SA22-7610-03

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 801.

Fourth Edition, September 2002

This is a major revision of SA22-7610-02.

This edition applies to Version 1 Release 4 of z/OS™ (5694-A01), to Version 1 Release 4 of z/OS.e (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Order documents through your IBM® representative or the IBM branch office serving your locality. Documents are not stocked at the address below.

IBM welcomes your comments. A form for readers’ comments may be provided at the back of this document, or you may address your comments to the following address:

International Business Machines Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY 12601-5400
United States of America

FAX (United States & Canada): 1+845+432-9405
FAX (Other Countries):
Your International Access Code +1+845+432-9405

IBMLink™ (United States customers only): IBMUSM10(MHVRCFS)
Internet e-mail: mhvrcfs@us.ibm.com
World Wide Web: http://www.ibm.com/servers/eserver/zseries/zos/webqs.html

If you would like a reply, be sure to include your name, address, telephone number, or FAX number.

Make sure to include the following in your comment or note:
- Title and order number of this document
- Page number or topic related to your comment

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

© Copyright International Business Machines Corporation 1988, 2002. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures vii

Tables ix

About this document xi

Summary of changes xv

Using the Services 1

ENFREQ — Listen For System Events 25

ENQ — Request Control of a Serially Reusable Resource 55

ESPIE — Extended SPIE 73

ESTAE and ESTAEX — Specify Task Abnormal Exit Extended 83

ETCON — Connect Entry Table 99

ETCRE — Create Entry Table 105

ETDEF — Create an Entry Table Descriptor (ETD) 109

ETDES — Destroy Entry Table 119

ETDIS — Disconnect Entry Table 123

EVENTS — Wait for One or More Events to Complete 127

EXTRACT — Extract TCB Information 133

FESTAE — Fast Extended STAE 139

FRACHECK — Check User's Authorization (for RACF Release 1.8.1 or earlier) 143

FREEMAIN — Free Virtual Storage 145

GETDSAB — Accessing the DSAB Chain 157

GETMAIN — Allocate Virtual Storage 165

GQSCAN — Extract Information From Global Resource Serialization Queue 179

GTRACE — GTF Trace Recording 191

HSPSERV — Read from and Write to a Hiperspace 201

IARR2V — Convert a Central Storage Address to a Virtual Storage Address 219

© Copyright IBM Corp. 1988, 2002

IARSUBSP — Create and Delete a Subspace 225

IARVSERV — Request to Share Virtual Storage 235

IARV64 — 64 Bit Virtual Storage Allocation 247

IAZXCTKN — Client Token Compare Service 295

IAZXJSAB — Obtain Information about a Currently Running Job 297

IEAARR — Establish an Associated Recovery Routine (ARR) 301

IEAFP — Floating Point Services 305

IEALSQRY — Linkage Stack Query 309

IEAMRMF3 — Obtain Address Space Dispatchability Data 313

IEAMSCHD — Schedule an SRB 321

IEANTCR — Create a Name/Token Pair 335

IEANTDL — Delete a Name/Token Pair 341

IEANTRT — Retrieve the Token from a Name/Token Pair 345

IEARBUP — RB Update Service 351

IEATDUMP — Transaction Dump Request 359

IEAVAPE — Allocate_Pause_Element 379

IEAVDPE — Deallocate_Pause_Element 383

IEAVPSE — Pause Service 387

IEAVRLS — Release 393

IEAVRPI — Retrieve_Pause_Element_Information Service 399

IEAVTPE — Test_Pause_Element Service 405

IEAVXFR — Transfer Service 409

IEECMDS — Query/Remove Attached Commands 415

IEEQEMCS — Query EMCS Console 425

IEEVARYD — Vary Device(s) Online and Offline 441

IEFPPSCN — Scan the Program Properties Table 453

IEFQMREQ — Invoke SWA Manager in Move Mode 461

IEFSSI — Dynamically Control a Subsystem 465

IEFSSVT — Create a Subsystem Vector Table 499

IEFSSVTI — Associate Function Routines with Function Codes 519

IOCINFO — Obtain MVS I/O Configuration Information 533

IOSADMF — Transfer HiperspaceData 539

IOSCAPF — Obtain the Actual Address of a Captured UCB 551

IOSCAPU — Capture, Release, or Obtain the Actual Address of a UCB 553

IOSCDR — Retrieve Configuration Data Records. 571

IOSCHPD — IOS CHPID Description Service 585

IOSCMXA — Obtain Address of the UCB Common Extension Segment 591

IOSCMXR — Obtain Address of the UCB Common Extension Segment 597

IOSCUMOD — IOS Control Unit Entry Build Service 599

IOSDCXR — Obtain Address of the Device Class Extension Segment 603

IOSENQ — IOS ENQ Service. 607

IOSINFO — Obtain the Subchannel Number for a UCB 613

IOSLOOK — Locate Unit Control Block 617

IOSPTHV — Validate I/O Paths 619

IOSUPFA — Obtain Address of the UCB Prefix Extension Segment . . . 629

IOSUPFR — Obtain Address of the UCB Prefix Extension Segment . . . 633

IOSWITCH — IOS Switch Information Service 635

IRDFSD — FICON Switch Data Services 641

IRDFSDU — FICON Switch Data Update Services 647

ISGLCRT — Create a Latch Set 651

ISGLOBT — Obtain a Latch 657

ISGLREL — Release a Latch. 663

ISGLPRG — Purge a Requestor from a Latch Set 669

ISGLPBA — Purge a Group of Requestors from a Group of Latch Sets 673

ITTFMTB — Generate Component Trace Format Table 677

ITZXFILT — Transaction Trace Filter Exit 681

IXGBRWSE — Browse/Read a Log Stream 685

IXGCONN — Connect/Disconnect to Log Stream 729

IXGDELET — Deleting Log Data from a Log Stream 751

IXGIMPRT — Import Log Blocks 767

IXGWRITE — Write Log Data to a Log Stream 781

Appendix. Accessibility 799

Notices 801

Index 805

Figures

1. Sample User Parameter List for Callers in AR Mode 5
2. Sample Macro Syntax Diagram 13
3. Continuation Coding 15
4. Return Code Area Used by ENQ 63
5. Characteristics and Restrictions for Standard Hiperspaces 203
6. Characteristics and Restrictions for ESO Hiperspaces. 209
7. RANGLIST and NUMRANGE Parameters 542

© Copyright IBM Corp. 1988, 2002

Tables

1. Passing User Parameters in AR Mode . . . 4
2. Sample Callable Service Syntax Diagram . . . 15
3. Service Summary . . . 17
4. Six-word Data Structure . . . 30
5. ENF Macro Event Codes . . . 32
6. Return Codes for the ENFREQ Macro . . . 45
7. Return Codes for the ENQ Macro with the RET=TEST Parameter . . . 64
8. Return Codes for the ENQ Macro with the RET=USE Parameter . . . 64
9. Return Codes for the ENQ Macro with the RET=CHNG Parameter . . . 65
10. Return Codes for the ENQ Macro with the RET=HAVE Parameter . . . 65
11. Return Codes for the ENQ Macro with the ECB Parameter . . . 67
12. Return Codes for the ESPIE TEST Macro . . . 81
13. Return and Reason Codes for the ESTAE Macro . . . 90
14. Return and Reason Codes for the ESTAEX Macro . . . 94
15. Return Code for the ETCON Macro . . . 101
16. Return Code for the ETCRE Macro . . . 107
17. Return Codes for the ETDES Macro . . . 121
18. Return Code for the ETDIS Macro . . . 125
19. Return Codes for the FESTAE Macro . . . 142
20. Return Codes for the FREEMAIN Macro . . . 152
21. Return Codes for the GETDSAB Macro . . . 160
22. Return and Reason Codes for the GETDSAB Macro . . . 160
23. Return Codes for the GETMAIN Macro . . . 176
24. Return Codes for the GQSCAN Macro . . . 185
25. Return Codes for the GTRACE TEST Macro . . . 193
26. Return Codes for the GTRACE DATA Macro . . . 197
27. Return and Reason Codes for HSPSERV SREAD and HSPSERV SWRITE . . . 207
28. Return and Reason Codes for HSPSERV CREAD and HSPSERV CWRITE . . . 213
29. Return and Reason Codes for the IARR2V Macro . . . 223
30. Storage Attributes Required for Subspaces . . . 229
31. Return and Reason Codes for the IARSUBSP Macro . . . 230
32. IARVSERV Permitted Storage Combinations . . . 236
33. Return and Reason Codes for the IARVSERV Macro . . . 241
34. Return and Reason Codes for the IARV64 Macro . . . 293
35. Return Codes for the IAZXCTKN Macro . . . 296
36. Return and Reason Codes for the IAZXJSAB Macro . . . 300
37. Return and Reason Codes for the IEAFP Macro . . . 307
38. Return Codes for IEALSQRY . . . 311
39. Return Codes for the IEAMRMF3 Macro . . . 316
40. Return Codes for the IEAMSCHD Macro . . . 332
41. Return Codes for the IEANTCR Macro . . . 338
42. Return Codes for the IEANTDL Macro . . . 343
43. Return Codes for the IEANTRT Macro . . . 348
44. Return and Reason Codes for the IEARBUP Macro . . . 356
45. Return and Reason Codes for the IEATDUMP Macro . . . 370
46. Return and Reason Codes for the IEEQEMCS Macro . . . 433
47. Return and Reason Codes for the IEEVARYD Macro . . . 444
48. Return Codes for the IEFPPSCN Macro . . . 456
49. Return Codes for the IEFQMREQ Macro . . . 462
50. Reason Codes for the IEFQMREQ Macro . . . 462
51. Return and Reason Codes for the IEFSSI Macro . . . 493
52. Return and Reason Codes for the IEFSSVT Macro . . . 515
53. Parameters Valid with IOSADMF Requests . . . 541

© Copyright IBM Corp. 1988, 2002

54. Return and Reason Codes for the IOSADMF Macro 543
55. Return and Reason Codes for the IOSCAPU CAPTUCB Macro 557
56. Return and Reason Codes for the IOSCAPU UCAPTUCB Macro 561
57. Return and Reason Codes for the IOSCAPU CAPTOACT Macro 566
58. Return and Reason Codes for the IOSCHPD Macro 589
59. ABEND Codes for the IOSENQ Macro 611
60. Return and Reason Codes for the IOSENQ Macro 611
61. Return Codes for the IOSLOOK Macro 618
62. Return and Reason Codes for the IOSWITCH Macro 638
63. Return Codes for IRDFSD macro 645
64. Return Codes for IRDFSDU macro. 650
65. ISGLCRT Return Codes. 653
66. ISGLOBT Return Codes. 661
67. ISGLREL Return Codes. 666
68. ISGLPRG Return Codes 671
69. ISGLPBA Return Codes. 676
70. Return and Reason Codes for the ITZXFILT Macro. 683
71. Return and Reason Codes for the IXGBRWSE Macro. 715
72. Return and Reason Codes for the IXGCONN Macro 739
73. Return and Reason Codes for the IXGDELET Macro 758
74. Return and Reason Codes for the IXGIMPRT Macro 772
75. Return and Reason Codes for the IXGWRITE Macro 788

About this document

This document supports z/OS (5694–A01) and z/OS.e (5655–G52).

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
- Residing in an APF-authorized library and link-edited with authorization code AC=1.

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 alphabetical 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 resides on an APF-authorized library.

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.

Assembler language programming is described in the following books:
- *HLASM Programmer's Guide*
- *HLASM Language Reference*

Using this book 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*.

© Copyright IBM Corp. 1988, 2002

Where to find more information

Where necessary, this document references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap* (GC28-1727).

The following table lists titles and order numbers for documents related to other products.

Short Title Used in This Document	Title	Order Number
*Principles of Operation**	*z/Architecture Principles of Operation*	SA22-7832
* Use the appropriate *Principles of Operation* document for the hardware you have installed.		
PSF/MVS System Programming Guide	*Print Services Facility/MVS System Programming Guide*	S544-3672

Accessing z/OS licensed documents on the Internet

z/OS licensed documentation is available on the Internet in PDF format at the IBM Resource Link™ Web site at:

`http://www.ibm.com/servers/resourcelink`

Licensed documents are available only to customers with a z/OS license. Access to these documents requires an IBM Resource Link user ID and password, and a key code. With your z/OS order you received a Memo to Licensees, (GI10-0671), that includes this key code. [1]

To obtain your IBM Resource Link user ID and password, log on to:

`http://www.ibm.com/servers/resourcelink`

To register for access to the z/OS licensed documents:

1. Sign in to Resource Link using your Resource Link user ID and password.
2. Select **User Profiles** located on the left-hand navigation bar.

Note: You cannot access the z/OS licensed documents unless you have registered for access to them and received an e-mail confirmation informing you that your request has been processed.

Printed licensed documents are not available from IBM.

You can use the PDF format on either **z/OS Licensed Product Library CD-ROM** or IBM Resource Link to print licensed documents.

Using LookAt to look up message explanations

LookAt is an online facility that allows you to look up explanations for most messages you encounter, as well as for some system abends and codes. Using LookAt to find information is faster than a conventional search because in most cases LookAt goes directly to the message explanation.

You can access LookAt from the Internet at:

1. z/OS.e™ customers received a Memo to Licensees, (GI10-0684) that includes this key code.

http://www.ibm.com/eserver/zseries/zos/bkserv/lookat/

or from anywhere in z/OS where you can access a TSO/E command line (for example, TSO/E prompt, ISPF, z/OS UNIX System Services running OMVS). You can also download code from the *z/OS Collection* (SK3T-4269) and the LookAt Web site that will allow you to access LookAt from a handheld computer (Palm Pilot VIIx suggested).

To use LookAt as a TSO/E command, you must have LookAt installed on your host system. You can obtain the LookAt code for TSO/E from a disk on your *z/OS Collection* (SK3T-4269) or from the **News** section on the LookAt Web site.

Some messages have information in more than one document. For those messages, LookAt displays a list of documents in which the message appears.

Information updates on the web

For the latest information updates that have been provided in PTF cover letters and Documentation APARs for z/OS and z/OS.e, see the online document at:

http://www.s390.ibm.com:80/bookmgr-cgi/bookmgr.cmd/BOOKS/ZIDOCMST/CCONTENTS

This document is updated weekly and lists documentation changes before they are incorporated into z/OS publications.

Summary of changes

Summary of changes
for SA22-7610-03
z/OS Version 1 Release 4

This document contains information previously presented in *z/OS MVS Programming: Authorized Assembler Services Reference ENF-IXG*, SA22-7610-02, which supports z/OS Version 1 Release 3.

New information

- Information is added to indicate this books supports z/OS.e.

This document contains terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

Starting with z/OS V1R2, you may notice changes in the style and structure of some content in this document—for example, headings that use uppercase for the first letter of initial words only, and procedures that have a different look and format. The changes are ongoing improvements to the consistency and retrievability of information in our documents.

Summary of changes
for SA22-7610-02
z/OS Version 1 Release 3

The document contains information previously presented in *z/OS MVS Programming: Authorized Assembler Services Reference ENF-IXG*, SA22-7610-01, which supports z/OS Version 1 Release 2.

New information

- A new qualifer is added to the ENF 56 event code.
- An appendix with z/OS product accessibility information has been added.

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes
for SA22-7610-01
z/OS Version 1 Release 2

The document contains information previously presented in *z/OS MVS Programming: Authorized Assembler Services Reference ENF-IXG*, SA22-7610-00, which supports z/OS Version 1 Release 1.

New information

- IEECMDS — Query/Remove Attached Commands
- IARV64 — 64–bit Virtual Storage Allocation
- IRDFSD — FICON Switch Data Services
- IRDFSDU — FICON Switch Data Update Services

Changed information

© Copyright IBM Corp. 1988, 2002

- “Using the Services” on page 1 has been changed to reflect 64–bit Virtual Storage addressing.
- GETDSAB — ″Accessing the DSAB Chain″ macro has been changed to include the new LOC parameter to specify whether the macro should examine all of the DSABs or only those which reside below the 16M line.
- IARR2V — ″Convert a Central Storage Address to a Virtual Storage Address″ has been chnaged to include the new VSA64 keyword.

Moved information

- ″IXGINVNT — Managing the LOGR Inventory Couple Data Set″ macro is now found only in MVS Programming: Assembler Services Reference

This document contains terminology, maintenance, and editorial changes, including changes to improve consistency and retrievability.

Summary of changes
for SA22-7610-00
z/OS Version 1 Release 1

The document contains information also presented in *MVS/ESA Programming: Authorized Assembler Services Reference Volume 2 (ENFREQ-ITTFMTB)*.

New information

- IOSCUMOD — IOS Control Unit Entry Build Service
- IOSENQ — IOS ENQ Service
- IOSWITCH — IOS Switch Information Service

Changed information

- IOSCHPD — IOS CHPID Description Service is updated.

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 book.

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.

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)” on page 2	Both
“Address Space Control (ASC) Mode” on page 3	Both
“ALET Qualification” on page 3	Both
“User Parameters” on page 4	Macros
“Telling the System about the Execution Environment” on page 5	Macros
“Specifying a Macro Version Number” on page 6	Macros
“Register Use” on page 7	Both
“Handling Return Codes and Reason Codes” on page 8	Both
“Handling Program Errors” on page 9	Both
“Handling Environmental and System Errors” on page 9	Both
“Using X-Macros” on page 10	Macros
“Macro Forms” on page 11	Macros
“Coding the Macros” on page 12	Macros
“Coding the Callable Services” on page 15	Callable Services
“Including Equate (EQU) Statements” on page 16	Callable Services
“Link-Editing Linkage-Assist Routines” on page 16	Callable Services
“Service Summary” on page 16	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; for example, support for access registers was introduced in MVS/ESA. Support for 64-bit registers was introduced in OS/390 R10. To take best advantage of the new architectures, some macros have more than one possible expansion. You are required to have the

© Copyright IBM Corp. 1988, 2002

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 via 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” on page 6.

Addressing Mode (AMODE)

A program can run in 24-bit, 31-bit, or 64-bit addressing mode. A program that executes in 24-bit or 31-bit addressing mode can invoke most of the services described in this book. A program that executes in 64-bit addressing mode has a smaller group of services that it can invoke.

In general,

- A program running in 24-bit addressing mode cannot pass parameters or parameter addresses that are higher than 16 megabytes. However, there are exceptions. For example, a program running in 24-bit addressing mode 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 running in 24-bit or 31-bit addressing mode cannot pass parameter addresses that are higher than 2 gigabytes, unless stated otherwise in the individual service description.
- If a program running in 31-bit or 64-bit addressing mode 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 64–bit addressing mode or 24–bit or 31–bit addressing mode. These macros check a global symbol set by the SYSSTATE macro. See “Telling the System about the Execution Environment” on page 5 for more information.

When you call a callable service in 24-bit or 31-bit addressing mode, 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 24-bit mode 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 below, but you may not 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, IDENTIFY, GTRACE, IARVSERV, LINKX, LOAD, MODESET, PHSER, POST, RESERVE, SDUMPX, SETRP, STAX, STIMER, STIMERM, STORAGE, SYNCHX, TIME, TIMEUSED, TTIMER, VRADATA, WAIT, WTO, WTOR, and XCTL.

You can invoke the following service in 64-bit addressing mode and may pass parameters and parameter addresses above 2 gigabytes: IARV64.

Before invoking a service in 64-bit addressing mode, you must inform system macros, by specifying SYSSTATE AMODE=64, that you are in 64-bit addressing mode. Only those options that result in calling the system by an SVC or PC may be invoked in 64-bit addressing mode. Any option that results in calling the system by a branch-entry may not be invoked in 64-bit addressing mode.

Unless explicitly stated otherwise, you should assume that a given service may not be invoked in 64-bit addressing mode and cannot accept parameters and parameter addresses above 2 gigabytes.

For information about 64-bit addressing mode 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” on page 5 for more information. Table 3 on page 17 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 book 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:

- All services allow you to place parameters in the primary address space.
- Some services *require* you to place parameters in the primary address space.
- Some services allow you to place parameters in any address space.

To identify where a service allows parameters to be located, 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 where, within the address space, the parameters reside.

The only ALETs that MVS services typically accept are:

- Zero (0), which specifies that the parameters reside 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 must not attempt to pass them to a service:

- One (1), which signifies that the parameters reside 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 book 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 book 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 to the parameters whose addresses are in the parameter list. Control parameters control the operation of the macro itself. User parameters are parameters that the 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.

Notes:

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 and ALETs. (See Figure 1 for the format of the list.)
ESTAEX	PARAM	SDWAPARM contains the address of an 8-byte area, which contains the address and ALET of the parameter list.

When a caller in AR mode 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 on page 5. 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 size

of the list. For example, Figure 1 shows the format of a list where an AR mode issuer of ATTACHX codes the PARAM parameter as follows:

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

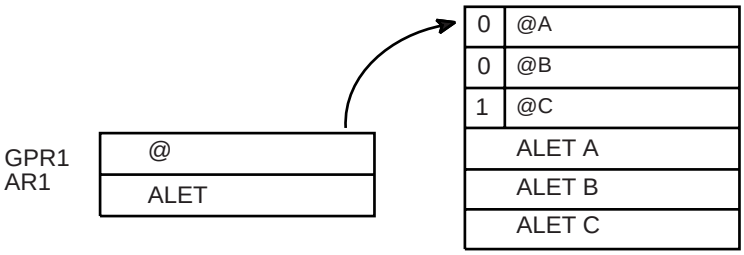


Figure 1. Sample User Parameter List for Callers in AR Mode

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 will run, 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 will run

For macros that are sensitive to their environment, you must use the SYSSTATE macro to define the environment. During the assembly stage, SYSSTATE sets one or more global symbols. Later, when the program runs, the macro checks the global symbols and generates the correct code, which might mean avoiding use of a z/Architecture instruction or using an access register. Table 3 on page 17 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. If you switch AMODE or ASC mode or to a different architecture code path, you should issue SYSSTATE immediately after the switch to indicate the new state. Without this information, the system assumes the macro is issued:

- In AMODE other than 64-bit
- In primary ASC mode
- In ESA/390 architectural level

The following table describes the relevant characteristics, the parameter on SYSSTATE, and the global symbol the macro checks.

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

Characteristic	Parameter on SYSSTATE	Global symbol
Architectural level of: • ESA/390 • ESA/390 but includes the ESA/390 architecture items required by OS/390 R10 • z/Architecture	ARCHLVL=0, 1 or 2	&SYSALVL

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 section. 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 performed a function similar to SYSSTATE. The SPLEVEL macro identifies the level of the operating system, so that a macro expansion can be tuned based on that level. This is used where macro expansions changed incompatibly. Because SPLEVEL applies to levels of the system no longer supported, it is not described in this section.

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 connection to a log stream, while version 2 adds new parameters in support of resource manager programs. Note that 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 new parameters are added 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 OS/390 and 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 also have the option of coding a specific version number using *plistver*, or of specifying MAX:

- *plistver* allows you 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.
- **MAX** allows you 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 in OS/390 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" section for the service provides that information. The section 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.

Note that the system uses ARs 0, 1, 14, and 15 as work registers for every service, regardless of whether the caller is in primary or AR address space control (ASC) mode. The system does not use ARs 2 through 13 for any service.

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 OS/390 version 2 release 10 or 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 book 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 book 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 section 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” on page 3. 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 book 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” on page 12.

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 book, each macro description includes a syntax table near the beginning of the macro description. The table shows how to code the macro. The syntax table does not explain the meanings of the parameters; the meanings are explained in the parameter descriptions that follow the syntax table.

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. To code the ICTL instruction, see *HLASM Language Reference*.

Figure 2 shows a sample macro, TEST, and summarizes all the coding information that is available for it. The table is divided into three columns, A, B, and C.

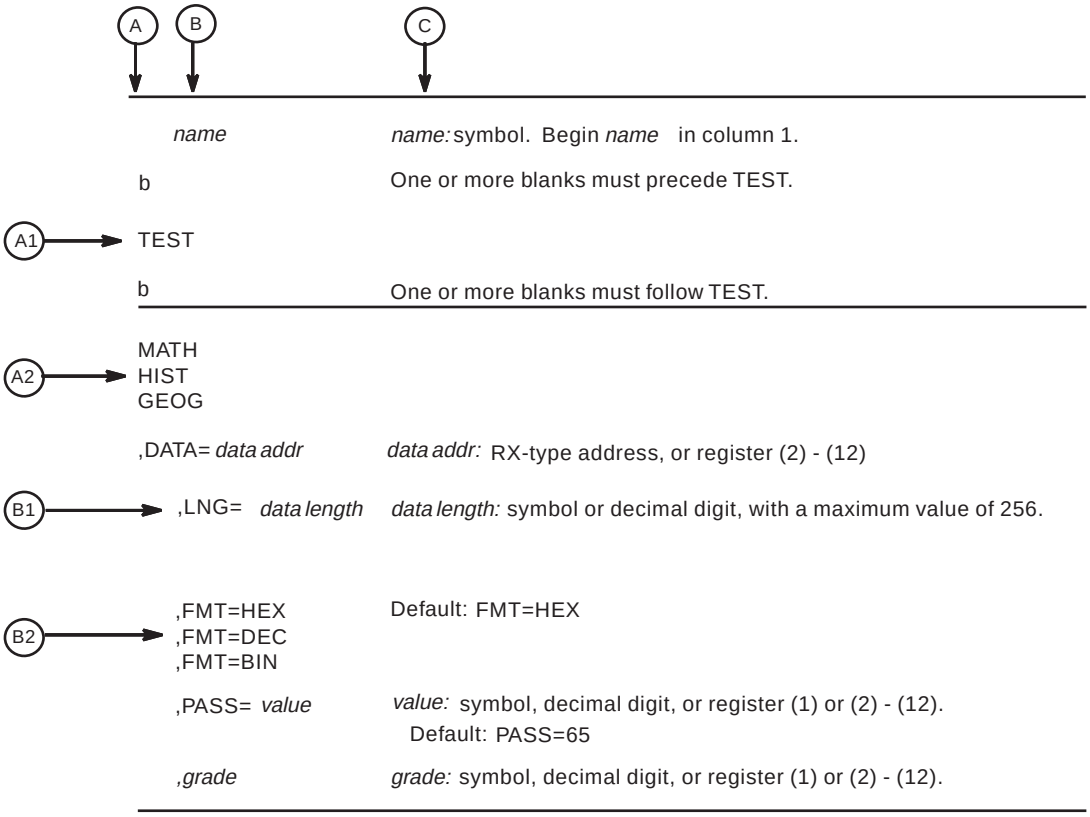


Figure 2. Sample Macro Syntax Diagram

- Column A and Column B contain those parameters that are allowed for the macro. Column A contains those parameters that are required; column B contains those parameters which are optional.
- If a single line appears, as shown in A1 and B1, then that is the only available choice for the particular parameter.
- If two or more lines appear together, 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.
- The third column, C, provides additional information about coding the macro.

When substitution of a variable is required in column 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.

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 italics.
- 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 third column before proceeding to the next parameter. The third 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 3 shows an example of each method.

```
1          10     16                                  44                                  72
NAME 1     OP1    OPERAND1,OPERAND2,OPERAND3,OPERAND4,OPERAND5,OPERAND6,OPX
                  ERAND7                      THIS IS ONE WAY
NAME 2     OP2    OPERAND1,OPERAND2           THIS IS ANOTHER WAY          X
                  OPERAND3,OPERAND4,                                       X
                  OPERAND5,OPERAND6,OPERAND7
```

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)
```

Table 2 shows the syntax diagram for the sample callable service SCORE.

Table 2. Sample Callable Service Syntax Diagram

```
                                  |
                                  |,(test_type
                                  |,level
CALL  SCORE                       |,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 3 on page 17 lists services described in the following:
- *z/OS MVS Programming: Authorized Assembler Services Reference ALE-DYN*
- *z/OS MVS Programming: Authorized Assembler Services Reference ENF-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 variable
- Whether the macro can be issued in 64-bit addressing mode.

Notes:

1. A program running in primary ASC mode when PASN=SASN=HASN 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	No	No
ATTACH	Yes (See note 1 on page 24)	No	Yes	No
ATTACHX	Yes	No	Yes	Yes
AXEXT	No	Yes	No	No
AXFRE	No	Yes	No	No
AXRES	No	Yes	No	No
AXSET	No	Yes	No	No
BPXEKDA	Yes	No	Yes	No
BPXESMF	Yes	No	Yes	No
CALLDISP	No	Yes	No	Yes
CALLRTM	No	Yes (See note 2 on page 24)	No	No
CHANGKEY	No	Yes	No	No
CIRB	No	No	No	No
CMDAUTH	No	No	No	No
COFCREAT	Yes	Yes	Yes	No
COFDEFIN	Yes	Yes	Yes	No
COFIDENT	Yes	Yes	Yes	No
COFNOTIF	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
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
CPF	No	No	No	No
CPOOL	No	Yes	No	No
CSRSI	No	Yes	No	No
CSRUNIC	Yes	Yes	No	No
CSVAPF	Yes (See note 11 on page 24)	Yes (See note 12 on page 24)	Yes	No
CSVDYNEX	Yes (See note 13 on page 24)	Yes (See note 14 on page 24)	Yes	No
CTRACE	No	No	Yes	No
CTRACECS	Yes	No	Yes	No
CTRACEWR	Yes	Yes	Yes	No
DATOFF	Yes	No	No	No
DEQ	No	No	No	Yes
DIV	Yes	No	Yes	No
DOM	No	No	No	Yes
DSPSERV	Yes	Yes	Yes	Yes
DYNALLOC	No	No	No	Yes
ENFREQ	No	No	No	No
ENQ	No	No	No	Yes
ESPIE	No	No	No	Yes
ESTAE (See note 3 on page 24)	No	No	Yes	No
ESTAEX	Yes	Yes	Yes	Yes
ETCON	No	Yes	No	No
ETCRE	No	Yes	No	No
ETDEF	Yes	Yes	No	No
ETDES	No	Yes	No	No
ETDIS	No	Yes	No	No
EVENTS	No	No	No	No
EXTRACT	No	No	No	No
FESTAE	No	No	No	No
FREEMAIN	Yes (See note 4 on page 24)	Yes	Yes	Yes
GETDSAB	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
GETMAIN	Yes (See note 4 on page 24)	Yes	Yes	Yes
GQSCAN	No	Yes	No	No
GTRACE	No	Yes	No	Yes
HSPSERV	Yes	Yes (See note 5 on page 24)	(See note 6 on page 24)	No
IARR2V	Yes	Yes	No	No
IARSUBSP	Yes	Yes	Yes	No
IARVSERV	Yes	Yes	Yes	No
IARV64	Yes	Yes	Yes	Yes
IAZXJSAB	Yes	Yes (See note 15 on page 24)	Yes	No
IEAARR	Yes	Yes	Yes	No
IEALSQRY	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
IEAVAPE	No	Yes	No	No
IEAVDPE	No	Yes	No	No
IEAVPSE	No	Yes	No	No
IEAVRLS	No	Yes	No	No
IEAVRPI	No	Yes	No	No
IEAVTPE	No	Yes	No	No
IEAVXFR	No	Yes	No	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
IOSADMF	No	No	Yes	No
IOSCAPF	No	Yes (See note 7 on page 24)	Yes	No
IOSCAPU	Yes	Yes (See note 7 on page 24)	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
IOSCDR	No	No	Yes	No
IOSCHPD	Yes	Yes	Yes	No
IOSCMXA	No	Yes (See note 7 on page 24)	Yes	No
IOSCMXR	No	Yes (See note 7 on page 24)	Yes	No
IOSDCXR	No	Yes (See note 7 on page 24)	Yes	No
IOSENQ	Yes	Yes	Yes	No
IOSINFO	No	No	No	No
IOSLOOK	No	No	No	No
IOSPTHV	No	No	Yes	No
IOSUPFA	No	Yes	Yes	No
IOSUPFR	No	Yes	Yes	No
IOSWITCH	Yes	Yes	Yes	No
ISGLCRT	No	Yes	N/A	No
ISGLOBT	No	Yes	N/A	No
ISGLREL	No	Yes	N/A	No
ISGLPRG	No	Yes	N/A	No
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
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

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
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
LLACOPY	No	No	Yes	No
LOAD	Yes	No	No	Yes
LOADWAIT	No	Yes	Yes	No
LOCASCB	Yes	Yes	Yes	No
LXFRE	No	Yes	No	No
LXRES	No	Yes	No	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	No	No	No	No
NMLDEF	No	No	No	No
NUCLKUP	No	No	No	No
OIL	No	No	No	No
OUTADD	No	No	No	No
OUTDEL	No	No	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
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 on page 24)	Yes (See note 8 on page 24)	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
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 on page 24)	Yes (See note 9 on page 24)	Yes	No
SDUMPX	Yes	Yes (See note 9 on page 24)	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
SYNCH	Yes (See note 1 on page 24)	No	Yes	No
SYNCHX	Yes	No	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
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 on page 24)	Yes	No	Yes
T6EXIT	No	No	No	No
UCBINFO	Yes	Yes	Yes	No
UCBLOOK	Yes	Yes	Yes	No
UCBPIN	Yes	Yes	Yes	No
UCBSCAN	Yes	Yes	Yes	No
VSMLIST	No	Yes	No	No
VSMLOC	No	Yes	No	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

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

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=SASN=HASN).

   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.

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 5 on page 32.

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 49 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:

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

© Copyright IBM Corp. 1988, 2002

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
14-15	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.

LISTEN Option

Syntax

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

---

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).
,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).
,QUAL=*qualifier*	*qualifier*: RX-type address or address in register (2) - (12). **Default:** QMASK=NONE
,BITQUAL=*bitqual*	*bitqual*: name of a 32-byte field, hexadecimal numeric value (X'xxx'), or address in register (2) - (12) **Default:** 32 bytes of X'00'.
,BITCOMPARE=SUBSET ,BITCOMPARE=INTERSECT ,BITCOMPARE=EQUAL	**Default:** BITCOMPARE=SUBSET
,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=*parm data*	*parm addr*: A-type address, or address in register (2) - (12). *parm data*: a fullword of data
,EOT=*NO* ,EOT=*YES*	**Default:** EOT=NO.
,EOM=*NO* ,EOM=*YES*	**Default:** EOM=NO.
,PLISTVER=2 ,PLISTVER=3 ,PLISTVER=MAX	**Default:** Version implied by keywords
,RELATED=(*value*)	*value*: Any text.
,XSYS=*NO* ,XSYS=*YES*	**Default:** XSYS=NO.

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 5 on page 32.

,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.

,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.

,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 5 on page 32 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 5 on page 32 for your event code. The mapping macro that defines symbolics possible for the qualifier also appears in Table 5 on page 32.

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. ENF ignores information in bytes 2 through 4 because QMASK=BYTE1.

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. 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.

BYTE1	First byte
BYTE2	Second byte
BYTE3	Third byte
BYTE4	Fourth byte
ALL	All four bytes
NONE	No bytes

,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 5 on page 32 for event code 51. 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. See the description of the QUAL and QMASK keywords in this section.

,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 will pass 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 will pass 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 will pass 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.

,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.

,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 will identify 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*.

Table 4 shows the six-word data structure pointed to by register 1 on entry to the listener user exit routine.

,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.

Table 4 shows the six-word data structure pointed to by register 1 on entry to the listener user exit routine.

Table 4. Six-word Data Structure

Address of parameter list supplied by the system for this event code
Fullword of zeroes
Fullword of data specified by the PARM parameter of the listen request that established the listen exit

Table 4. Six-word Data Structure (continued)

Fullword of zeroes
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 on 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 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. 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
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.

Event Codes

Table 5 lists the event codes and their corresponding QUAL values.

Table 5. ENF Macro Event Codes

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.	The defined QUAL values is: **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 SI00PCCAxxx 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

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
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*. For more information about a DDR swap, see *z/OS MVS Recovery and Reconfiguration Guide*.	None	8-byte parameter list. The first four bytes contain the address of the UCB for the device to which the volume was swapped. The second four bytes contain the address of the UCB for the device from which the volume was swapped.	EXIT / NO

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
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 5. 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.	**BYTE 1** Device class (Byte 3 from UCBTYP) **BYTE 2** Reserved **BYTES 3-4** Qualifier number Each qualifier number designates a type of change, such as I/O subchannel change, device available, or a configuration data record (CDR) 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.	Mapped by IOSDDACH	EXIT / NO

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
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. 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 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
36	The system wrote a record to the logrec data set or the logrec log stream. ENF passes to the listener user exit routine a parameter list containing the record information. See IFBENF36 in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)* for details about the contents of the parameter list. 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'9x'.	Mapped by IFBENF36	EXIT or SRBEXIT / NO

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
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 For information about these accounting-related events, see *z/OS MVS System Management Facilities (SMF)*.	• ENF37Q00 - SMF was initialized • ENF37Q01 - SMF ended • ENF37Q02 - SMF INTVAL parameter changed • ENF37Q03 - SMF SYNCVAL parameter changed • ENF37Q04 - SMF interval expired • ENF37Q05 - SMF interval sync processing disabled	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.	• ARENEVENTREG - IXCARM macro with the REGISTER parameter was issued. • ARENEVENTREADY - IXCARM macro with the READY parameter was issued. • ARENEVENTDEREG - IXCARM macro with the DEREGISTER parameter was issued (or the system deregistered an element because of an error). • ARENEVENTCDSCONNECT - This system has acquired (or regained) access to the automatic restart management couple data set.	Mapped by IXCYAREN	SRBEXIT / NO

Table 5. 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, Vol 2 (DCCB-ITZYRETC)*.	• ENF40_INIT - A JES completed initialization • ENF40_TERM - A JES ended normally	Mapped by IEFENF40	EXIT / NO

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
41	A WLM event occurred. The following qualifiers for ENF code 41 are provided: 1. BYTE1 1 - Policy change initiated, current mode is goal 2 - Policy change completed, current mode is goal 3 - Policy change failed, current mode is goal 2. BYTE2 1 - Policy change initiated, current mode is compatibility 2 - Policy change completed, current mode is compatibility 3 - Policy change failed, current mode is compatibility 3. BYTE3 1 - Workload activity reporting failed and has begun recovery 2 - Workload activity reporting recovery was successful 3 - Workload activity reporting recovery was not successful 4. BYTE4 1 - WLM service definition was successfully installed	WLMENF11 - A VARY WLM,POLICY= command was issued while in goal mode WLMENF12 - A VARY WLM,POLICY= command completed while in goal mode WLMENF13 - A VARY WLM,POLICY= command failed while in goal mode. The new policy could not be activated on this system. WLMENF21 - A VARY WLM,POLICY= command was issued while in compatibility mode WLMENF22 - A VARY WLM,POLICY= command completed while in compatibility mode WLMENF23 - A VARY WLM,POLICY= command failed while in compatibility mode. 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

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
42	An SRM event occurred. The following qualifiers for code 42 are provided: 1. BYTE1 1 - Mode change initiated, current mode is goal 2 - Mode change complete, current mode is compatibility 3 - Mode change failed, current mode is goal 4 - Mode change initiated, current mode is compatibility 5 - Mode change complete, current mode is goal 6 - Mode change failed, current mode is compatibility 2. BYTE2 1 - SET IPS change initiated 2 - SET IPS change completed 3 - SET IPS change failed 4 - SET ICS change initiated 5 - SET ICS change completed 6 - SET ICS change failed	SRMENF11 - A MODIFY WLM,MODE=COMPAT command was issued SRMENF12 - A MODIFY WLM,MODE=COMPAT command completed SRMENF13 - A MODIFY WLM,MODE=COMPAT command failed SRMENF14 - A MODIFY WLM,MODE=GOAL command was issued SRMENF15 - A MODIFY WLM,MODE=GOAL command completed SRMENF16 - A MODIFY WLM,MODE=GOAL command failed SRMENF21 - A SET IPS command was issued SRMENF22 - A SET IPS command completed SRMENF23 - A SET IPS command failed SRMENF24 - A SET ICS command was issued SRMENF25 - A SET ICS command completed SRMENF26 - A SET ICS command failed	Mapped by IRARENF1	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

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
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	OS/390 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. See *z/OS MVS Programming: Assembler Services Guide* for a description of using ENF event 48 for system logger and *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)* for a description of the events mapped by the IXGENF macro.	None	Mapped by IXGENF	SRBEXIT / YES
49	The logrec output recording medium has been changed by the SETLOGRC command.	None	IFBNTASM	SRBEXIT / NO

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
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 be able to 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.	The qualifier (QUAL parameter) has the following format: BYTE1 Type of signal information x'01' Contention data x'02' RNL changes x'03' Mode changes BYTE2 Always x'00' BYTE3 Varies with event BYTE4 Varies with event The defined QUAL values are: **Qualifier** — **Information type** x'01000001' — Local resource contention x'01000002' — Global resource contention x'01000003' — Contention related recovery information x'02000001' — User job suspended due to RNL change x'02000002' — User job resumed following RNL change x'0300yyzz' — GRS mode changes zz = new mode yy = old 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

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
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
55	SRM has detected a significant MVS image event, which is being signalled. The qualifiers and parameters further define the event.	ENF55QLF_SHORTAGE_ RECOGNIZED (X'80000000') - SRM has recognized a storage shortage caused by an excessive number of fixed frames. ENF55QLF_SHORTAGE_ RELIEVED (X'40000000') - Storage shortage due to excessive fixed storage relieved.	Mapped by IRAENF55	EXIT or SRBEXIT / NO
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

Table 5. ENF Macro Event Codes (continued)

Event Code	Description	Qualifier	Parameter List Passed to the User Exit	Exit Type / Cross-System Capable
58	The state of a SYSOUT data set has changed. The SYSOUT data set was dynamically allocated using the DALRTCTK text unit. For more information, see the "Listening for Events" section of the "JES Client/Server Print Interface" chapter in *z/OS MVS Using the Subsystem Interface*.	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.	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

Return Codes

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

Table 6. Return Codes for the ENFREQ Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: ENFREQ processing completed successfully. **Action**: None

Table 6. Return Codes for the ENFREQ Macro (continued)

Hexadecimal Return Code	Meaning and Action
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.
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 Listn 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.

Table 6. Return Codes for the ENFREQ Macro (continued)

Hexadecimal Return Code	Meaning and Action
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, due to 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.

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:

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

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 5 on page 32.

,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 45.

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:

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.
,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 ,BITCOMPARE=INTERSECT ,BITCOMPARE=EQUAL	**Default:** BITCOMPARE=SUBSET

,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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ENFREQ.
ENFREQ	
b	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).
,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 ,BITCOMPARE=INTERSECT ,BITCOMPARE=EQUAL	**Default:** BITCOMPARE=SUBSET
,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* ,EOT=*YES*	**Default:** EOT=NO.
,EOM=*NO* ,EOM=*NO*	**Default:** EOM=YES.
,PLISTVER=2 ,PLISTVER=3 ,PLISTVER=MAX	**Default:** Version implied by keywords
,RELATED=*(value)*	*value*: Is any text.
,XSYS=*NO* ,XSYS=*YES*	**Default:** XSYS=NO.
,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:

---

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=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:

---

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

ENFREQ	
b	One or more blanks must follow ENFREQ.

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.

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 the section on 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 on using the ENQ macro to serialize resources, see the *z/OS MVS Programming: Authorized Assembler Services Guide*.

© Copyright IBM Corp. 1988, 2002

Environment

The requirements for callers of ENQ are:

Minimum authorization:	Problem state with any PSW key. For the SMC, ECB, TCB, MASID, and MTCB parameters or when the specified *qname* is SYSZ*, SYSCTLG, SYSDSN, SYSIEA01, SYSIEECT, SYSIEFSD, SYSIGGV1, SYSIGGV2, SYSPSWRD, SYSVSAM, or SYSVTOC. 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: PASN=HASN=SASN or 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:	Control parameters must be in the primary address space. Except for the TCB, all parameters can reside above 16 megabytes.

Programming Requirements

None.

Restrictions

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.

---

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).
, ,*rname addr*	*rname addr*: A-type address or register (2) - (12).
, ,E ,S	**Default**: E
,	

,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 ,SYSTEM ,SYSTEMS	**Default**: STEP
)	
,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	**Default**: RET=NONE
,SMC=NONE ,SMC=STEP	**Default**: SMC=NONE
,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.
,MTCB=*matching-tcb addr*	*matching-tcb addr*: A-type address or register (2) - (12). **Note**: MASID is required with MTCB.
,RNL=YES ,RNL=NO	**Default**: RNL=YES
,RELATED=*value*	*value*: any valid macro keyword specification.
,LINKAGE=SVC ,LINKAGE=SYSTEM	**DEFAULT**: LINKAGE=SVC

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.

,
,*rname addr*
Specifies the address of the name used together with *qname* to represent a single resource. The name must be from 1 to 255 bytes long, can be qualified, and can contain any valid hexadecimal character.

,
,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 or not you also specify MASID and MTCB:

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

NONE Control of all the resources is unconditionally requested.

See “Return and Reason Codes” on page 63 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 *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
	4	off
	8	off
	14	off
RET=HAVE	0	on
	8	on
	14	off
	18	off
RET=TEST	0	off
	4	off
	8	off
	14	off
RET=USE	0	on
	4	off
	8	off
	14	off
	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 addressible 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. 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. When RNL=NO is specified the ENQ request will be ignored by alternative serialization products. Refer to *z/OS MVS Planning: Global Resource Serialization*, RNL Processing, 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 4.

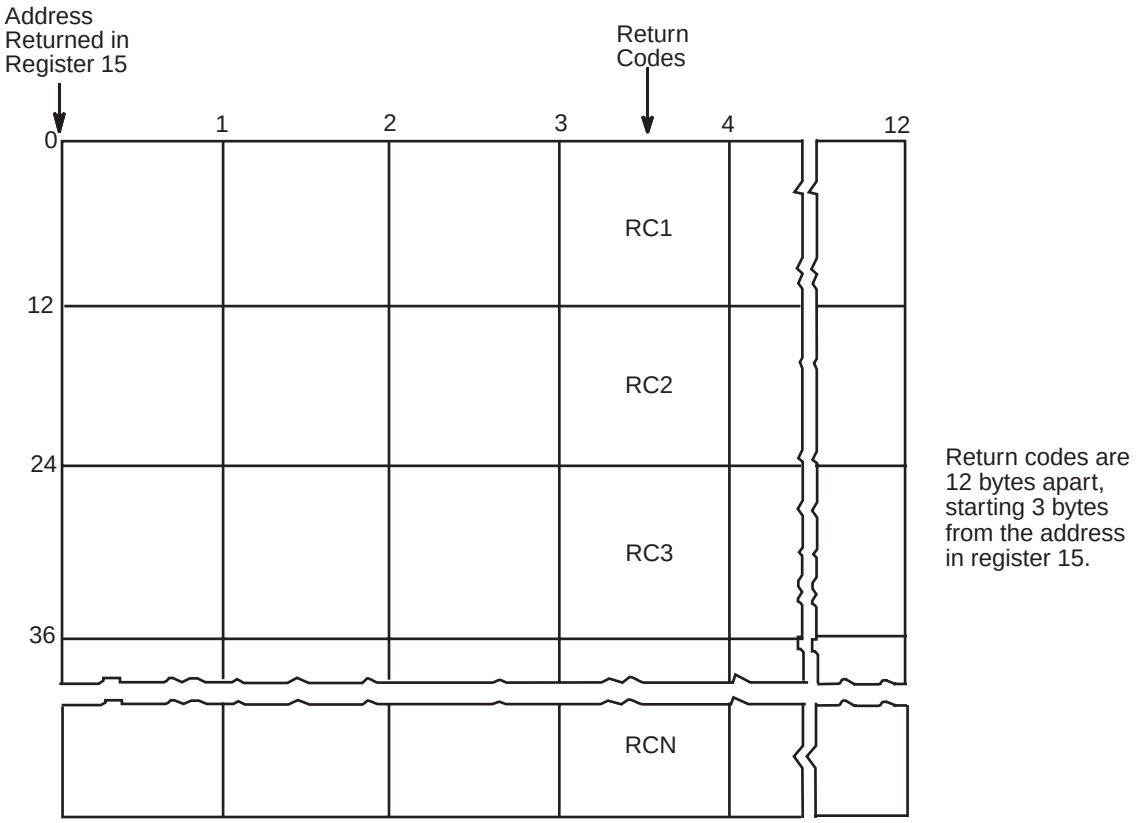


Figure 4. Return Code Area Used by ENQ

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 7 on page 64.

Table 7. 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 8.

Table 8. 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.

Table 8. Return Codes for the ENQ Macro with the RET=USE Parameter (continued)

Hexadecimal Return Code	Meaning and Action
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 9.

Table 9. 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.
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 10.

Table 10. 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.

Table 10. Return Codes for the ENQ Macro with the RET=HAVE Parameter (continued)

Hexadecimal Return Code	Meaning and Action
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.
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.
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 11.

Table 11. 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.

Table 11. Return Codes for the ENQ Macro with the ECB Parameter (continued)

Hexadecimal Return Code	Meaning and Action
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.
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:

---

name *name*: symbol. Begin *name* in column 1.

b	One or more blanks must precede ENQ.
ENQ	
b	One or more blanks must follow ENQ.

(

qname addr	*qname addr*: A-type address or register (2) - (12).
,	
,*rname addr*	*rname addr*: A-type address or register (2) - (12).
, ,E ,S ,	**Default**: E
,*rname length*	*rname length*: symbol or decimal digit. **Default**: assembled length of *rname*
, ,STEP ,SYSTEM ,SYSTEMS	**Default**: STEP

)

,RET=CHNG ,RET=HAVE ,RET=TEST ,RET=USE ,RET=NONE	**Default**: RET=NONE
,SMC=NONE ,SMC=STEP	**Default**: SMC=NONE
,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.

,RNL=YES ,RNL=NO	**Default**: RNL=YES
,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:

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).

,	
,*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.
,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.

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 sections 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*, with the exception of interruption type 17. This interruption type designates page faults, and its use is restricted to programs that are APF-authorized or run in PSW key 0 - 7.

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 headings 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 sections 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:

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. To specify page fault processing, the caller must be APF-authorized.

© Copyright IBM Corp. 1988, 2002

Dispatchable unit mode:	Task
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:

---

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.

SET	
,*exit addr*	*exit addr*: A-type address or register (2) - (12).
,*(interruptions)*	*interruptions*: Decimal numbers 1 - 15 or 17 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
17	Page fault

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 (except for type 17) 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 EPIE 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 1

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
```

Example 2

Give control to the exit routine located at EXIT when a page fault occurs.

```
ESPIE SET,EXIT,(17)
```

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	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 or 17 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.
,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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ESPIE.
ESPIE	
b	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.
,(interruptions)	*interruptions*: Decimal number 1 - 15 or 17 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:

---

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:

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 12. 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.
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
```

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 abnormally ending.

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. For information about how to select a macro for an MVS/SP version other than the current version, see "Compatibility of MVS Macros" on page 1.

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. For recovery routines defined through the ESTAEX macro, the AMODE will always be 31-bit. The AMODE at the retry point will be the same as the AMODE on entry to the recovery routine.

Callers that are in primary address space control (ASC) mode and not in cross memory mode (the primary, secondary, and home address spaces are the same) can issue either ESTAE or ESTAEX. Callers that are in access register (AR) mode or in cross memory mode (the primary, secondary, and home address spaces are different) 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 or your program requires a branch entry. In these cases, you should 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 via 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 section on providing recovery in *z/OS MVS Programming: Authorized Assembler Services Guide* for information about writing recovery routines.

The descriptions of ESTAE and ESTAEX in this book 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.

© Copyright IBM Corp. 1988, 2002

Note

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

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key. To use the CANCEL, BRANCH, KEY, RECORD, 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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ESTAE.
ESTAE	
b	One or more blanks must follow ESTAE.

---

exit addr 0	*exit addr*: A-type address, or register (2) - (12).
,CT ,OV	**Default**: CT
,PARAM=*list addr*	*list addr*: A-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	**Default**: XCTL=NO
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	**Default**: PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default**: ASYNCH=YES
,CANCEL=YES ,CANCEL=NO	**Default**: CANCEL=YES

,TERM=NO ,TERM=YES	**Default**: TERM=NO
,BRANCH=NO ,BRANCH=YES,SVEAREA=*save addr*	**Default**: BRANCH=NO *save addr*: A-type address, or register (2) - (12) or (13).
,KEY=SAVE ,KEY=*storage key*	*storage key*: Any numeral in the range 0-15.
,RECORD=NO ,RECORD=YES	**Default**: RECORD=NO
,TOKEN=*token addr*	*token addr*: A-type address, or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SDWALOC31=NO ,SDWALOC31=YES	**Default**: SDWALOC31=NO

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.

Notes:

1. You should understand PURGE processing before using this parameter. For information on PURGE processing, see *z/OS DFSMSdfp Advanced Services*.
2. When using PURGE, you should consider any access-method ramifications. See the appropriate DFP manual 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 routine 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.

Usage Notes:

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.
- OS/390 UNIX is canceled and the user's task is in a wait in the OS/390 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.

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 13. 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.
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.

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

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 – Therecovery 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),PARM=(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:

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ESTAEX.
ESTAEX	
b	One or more blanks must follow ESTAEX.

---

exit addr 0	*exit addr*: A-type address, or register (2) - (12).
,CT ,OV	**Default**: CT
,PARAM=*list addr*	*list addr*: A-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES	**Default**: XCTL=NO
,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	**Default**: PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default**: ASYNCH=YES
,CANCEL=YES ,CANCEL=NO	**Default**: CANCEL=YES
,TERM=NO ,TERM=YES	**Default**: TERM=NO
,RECORD=NO ,RECORD=YES	**Default**: RECORD=NO
,TOKEN=*token addr*	*token addr*: A-type address, or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

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 return and reason codes for ESTAEX are listed below.

Table 14. 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.
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.

Table 14. Return and Reason Codes for the ESTAEX Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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.
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.

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ESTAE or ESTAEX.
ESTAE ESTAEX	
b	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 ,PURGE=QUIESCE ,PURGE=HALT	**Default**: PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default**: ASYNCH=YES
,CANCEL=YES ,CANCEL=NO	**Default**: CANCEL=YES
,TERM=NO ,TERM=YES	**Default**: TERM=NO
,RECORD=NO ,RECORD=YES	**Default**: RECORD=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SDWALOC31=NO ,SDWALOC31=YES	**Default**: SDWALOC31=NO
,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, and TOKEN 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.

Syntax

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

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ESTAE or ESTAEX.
ESTAE ESTAEX	
b	One or more blanks must follow ESTAE or ESTAEX.

---

exit addr 0 ,CT ,OV	*exit addr*: RX-type address, or register (2) - (12).
,PARAM=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,XCTL=NO ,XCTL=YES ,PURGE=NONE ,PURGE=QUIESCE ,PURGE=HALT	
,ASYNCH=YES ,ASYNCH=NO	
,CANCEL=YES ,CANCEL=NO	**Default**: CANCEL=YES
,TERM=NO ,TERM=YES	
,BRANCH=NO	**Note**: BRANCH and SVEAREA are not valid on ESTAEX.

,BRANCH=YES,SVEAREA=*save addr*	*save addr*: RX-type address, or register (2) - (12) or (13).
,KEY=SAVE ,KEY=*storage key*	*storage key*: Any numeral in the range 0-15. **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 ,TKNPASS=YES	**Default**: TKNPASS=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SDWALOC31=NO ,SDWALOC31=YES	**Default**: SDWALOC31=NO
,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.

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:

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.

© Copyright IBM Corp. 1988, 2002

Input Register Information

Register 13 must point to a standard register save area that must be addressable in primary mode.

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:

---

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.

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*
Specifies the address of a list of linkage index 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 indexes 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.

,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 15. 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:

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

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

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:

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, Vol 2 (DCCB-ITZYRETC)*.

Restrictions

None.

Input Register Information

Before issuing the ETCRE macro, the caller must ensure that general purpose register 13 points to a standard 72-byte save area addressable in primary mode.

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

© Copyright IBM Corp. 1988, 2002

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:

---

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

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 zeroes, 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 16. 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
```

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 128 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:

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 should 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 section 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

© Copyright IBM Corp. 1988, 2002

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, Vol 2 (DCCB-ITZYRETC)* 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

This form is described as follows:

---

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 TYPE=ENTRY	**Valid Parameters**: RELATED **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.
,ARR=*arr*	*arr*: A-type address, or alphanumeric character string enclosed by single quotation marks.
,ASYNCH=YES ,ASYNCH=NO	**Default**: ASYNCH=YES Valid only when ARR is also coded.

,CANCEL=YES ,CANCEL=NO	**Default**: CANCEL=YES Valid only when ARR is also coded.
,ASCMODE=PRIMARY ,ASCMODE=AR	**Default**: ASCMODE=PRIMARY
,EAX=*eax-value*	*eax-value*: Half-word decimal digit.
,EK=*entry-key*	*entry-key*: Decimal digit 0 - 15.
,EKM=*key-list* **Note**: EKM is required with PKM=REPLACE.	*key-list*: List of keys or key ranges where a key is a number 0 - 15.
,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 ,PC=BASIC	**Default**: PC=STACKING
,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 ,PKM=REPLACE	**Default**: PKM=OR
,RAMODE=31 ,RAMODE=24	**Default**: RAMODE=31
,RELATED=*value*	*value*: Any valid macro parameter specification.
,SASN=OLD ,SASN=NEW	**Default**: SASN=OLD
,SSWITCH=NO ,SSWITCH=YES	**Default**: SSWITCH=NO
,STATE=PROBLEM ,STATE=SUPERVISOR	**Default**: STATE=PROBLEM

Parameters

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.

,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
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 an uninitialized storage area (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

The form of SET is described as follows:

---

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

b	One or more blanks must precede ETDEF.
ETDEF	
b	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.
,ASYNCH=YES ,ASYNCH=NO	**Default**: ASYNCH=YES Valid only when ARR is also coded.
,CANCEL=YES ,CANCEL=NO	**Default**: CANCEL=YES Valid only when ARR is also coded.
,ASCMODE=PRIMARY ,ASCMODE=AR	**Default**: ASCMODE=PRIMARY
,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 ,PC=BASIC	**Default**: PC=STACKING
,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 ,PKM=REPLACE	**Default**: PKM=OR
,RAMODE=31 ,RAMODE=24	**Default**: RAMODE=31
,RELATED=*value*	*value*: Any valid macro parameter specification.
,SASN=OLD ,SASN=NEW	**Default**: SASN=OLD
,SSWITCH=NO ,SSWITCH=YES	**Default**: SSWITCH=NO
,STATE=PROBLEM ,STATE=SUPERVISOR	**Default**: STATE=PROBLEM

Parameters

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

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.

,NUMETE=*nbr-of-entries*
Specifies the number of contiguous entries in the ETD. *nbr-of-entries* is a decimal value from 1 to 128. 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.

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, due to 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.

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:

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

Before issuing the ETDES macro, the caller must ensure that general purpose register 13 points to a standard 72-byte save area addressable in primary mode.

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.

© Copyright IBM Corp. 1988, 2002

Syntax

The ETDES macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ETDES.
ETDES	
b	One or more blanks must follow ETDES.

TOKEN=*addr*	*addr*: RX-type address or register (0) - (12).
,PURGE=NO ,PURGE=YES	**Default**: PURGE=NO
,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 17. 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:

---

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 ,PURGE=YES	**Default**: PURGE=NO
,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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ETDES.
ETDES	
b	One or more blanks must follow ETDES.

TOKEN=*addr*	*addr*: RX-type address or register (0) - (12).
,PURGE=NO ,PURGE=YES	**Default**: PURGE=NO
,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.

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:

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

Before issuing the ETDIS macro, the caller must ensure that general purpose register 13 points to a standard 72-byte save area addressable in primary mode.

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

© Copyright IBM Corp. 1988, 2002

Performance Implications

None.

Syntax

The ETDIS macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ETDIS.
ETDIS	
b	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 18. 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*.

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.

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.

For information about how to use this macro on an MVS/SP version other than the current version, see “Compatibility of MVS Macros” on page 1.

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:

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:

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
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.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede EVENTS.
EVENTS	
b	One or more blanks must follow EVENTS.

ENTRIES=*n* ENTRIES=*addr*	*n*: Decimal digits 1-32767 *addr*: Register (2) - (12).
ENTRIES=DEL,TABLE=*tab addr* TABLE=*tab addr*	*tab addr*: Symbol, RX-type address, or register (2) - (12). **Note**: If the ENTRIES parameter is specified as indicated in the first two formats, no other parameters may be specified.
,ECB=*ecb addr* ,LAST=*last addr*	*ecb addr*: Symbol, RX-type address, or register (2) - (12). *last addr*: Symbol, RX-type address, or register (2) - (12). **Note**: If LAST is specified, WAIT must also be specified.
,WAIT=YES ,WAIT=NO	**Note**: Do not specify WAIT=YES when running in a disabled state.
,BRANCH=NO ,BRANCH=YES	**Default**: BRANCH=NO

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.

Notes:

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.

Notes:

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.

Notes:
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
                   ST          1,TABLE

        NEXTREC    GET         TPDATA,KEY
                   ENQ         (RESOURCE,ELEMENT,E,,SYSTEM)
                   READ        DECBRW,KU,,'S',MF=E
                   LA          3,DECBRW
                   EVENTS      TABLE=TABLE,ECB=(3),WAIT=YES

                   WRITE       DECBRW,K,MF=E
                   LA          3,DECBRW
        RETEST     EVENTS      TABLE=TABLE,ECB=(3),WAIT=NO
                   LTR         1,1
                   BNZ         NEXTREC

                   B           RETEST

        TABLE      DS          F
```

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.

Notes:

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:

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:

© Copyright IBM Corp. 1988, 2002

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede EXTRACT.
EXTRACT	
b	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 via LOGON)
- In a background TSO/E session (initiated via the TSO/E TMP, IKJEFT01).
- In a TSO/E environment initialized outside of the TSO/E TMP (initiated via the IKJTSOEV service).

TJID Is the address space identifier (ASID) for a foreground TSO/E session (initiated via 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'338'. 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:

---

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.
,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:

---

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*: RX-type address, or register (2) - (12).
,'S' *,tcb addr*	*tcb addr*: RX-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
,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.

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.

For information about how to use this macro on an MVS/SP version other than the current version, see “Compatibility of MVS Macros” on page 1.

The FESTAE macro expansion has no external linkage.

Environment

The requirements for the caller are:

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 MVS/XA 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.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FESTAE.
FESTAE	
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 ,XCTL=YES	**Default**: XCTL=NO
,PURGE=NONE ,PURGE=HALT ,PURGE=QUIESCE	**Default**: PURGE=NONE
,ASYNCH=YES ,ASYNCH=NO	**Default**: ASYNCH=YES
,TERM=NO ,TERM=YES	**Default**: TERM=NO
,RECORD=NO ,RECORD=YES	**Default**: RECORD=NO
,ERRET=*label*	*label*: Any valid assembler name.
,SDWALOC31=NO ,SDWALOC31=YES	**Default**: SDWALOC31=NO

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 19. 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
```

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.

© Copyright IBM Corp. 1988, 2002

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:

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.

© Copyright IBM Corp. 1988, 2002

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.

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 section 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede FREEMAIN.
FREEMAIN	
b	One or more blanks must follow FREEMAIN.

LC,LA=*length addr* LU,LA=*length addr* L,LA=*length addr* VC	*length addr*: A-type address, or register (2) - (12).

VU	
V	
EC,LV=*length value*	*length value*: symbol, decimal number, or register (2) - (12).
EU,LV=*length value*	
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.

Notes:

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*.

Notes:

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” on page 147 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 20. 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede FREEMAIN.
FREEMAIN	
b	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. **Notes:** 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.
,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:

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* ,LV=*length value*	*length addr*: RX-type address or register (2) - (12). *length value*: symbol, decimal number, or register (2) - (12). **Notes:** 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.

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:

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).

GETDSAB can be used only by application programs that are the only task running in an address space, for either nonconcatenated DD statements or data sets that reside on a single volume.

The GETDSAB service does not provide a recovery environment. Because the service runs in task mode, the system will use any recovery environment defined to the caller prior to invoking GETDSAB.

Restrictions

There are no restrictions or limitations for using this macro.

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

© Copyright IBM Corp. 1988, 2002

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.

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede GETDSAB.
GETDSAB	
b	One or more blanks must follow GETDSAB.

FIRST NEXT DCBPTR=*dcb addr* DDNAME=*dd addr*	*dcb addr*: RX-type address, or register (2) - (12). *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** ,LOC=ANY	**Default** = BELOW

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.

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 16Mb line

LOC=ANY searches both below and above the 16Mb 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 due to input error. The DDNAME specified or obtained was all blanks.
12	1210	Request failed due to input error. TCBPTR was specified when DCBPTR points to an open DCB.
12	1220	Request failed due to input error. The DSAB pointed to by DSABPTR is not valid.
12	1230	Request failed due to 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
```

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.

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede GETDSAB.
GETDSAB	
b	One or more blanks must follow GETDSAB.

MF=(L,*stor addr*) MF=(L,*stor addr,attr*)	*stor addr*: symbol. *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.

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede GETDSAB.

GETDSAB

b	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** ,LOC=ANY	**Default** = BELOW
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.

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 section 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: 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 specified.

  The caller can specify CHECKZERO=YES to detect these and other cases where the system clears the requested storage to zeros.

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.

© Copyright IBM Corp. 1988, 2002

Environment

The requirements for the caller are:

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.

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 MVS-recognized state of disablement, which can be attained by obtaining the CPU lock.

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.
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.
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 releasing private storage. If your program is not running in cross-memory mode, setting GPR 4 to 0 identifies the input TCB as the TCB of the current task. If your program is running in cross-memory mode, 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 section 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:** In an AMODE 64 routine, a successful RC or RU 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:** In an AMODE 64 routine, 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.
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.
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:

---

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* 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*	*length addr*: A-type address, or register (2) - (12). *length value*: symbol, decimal number, or register (2) - (12). If RC or RU is specified, register (0) may also be specified. *addr*: A-type address or register (2) - (12). **Note:** RC, RU, VRC, or VRU must be used for address greater than 16 megabytes.
RC,LV=*length value* RU,LV=*length value* 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 ,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)	**Default**: LOC=RES **Note**: You must specify the INADDR parameter with EXPLICIT.
,INADDR=*stor addr*	*stor addr*: RX-type address or register (1)-(12). **Note:** This parameter can only be specified with LOC=EXPLICIT.
,OWNER=HOME ,OWNER=PRIMARY ,OWNER=SECONDARY ,OWNER=SYSTEM	**Default:** OWNER=HOME
,CHECKZERO=YES ,CHECKZERO=NO	**Default:** 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:

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)*

Specifies the type of GETMAIN request:

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.

Notes:

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 section about selecting the right subpool for virtual storage requests in *z/OS MVS Programming: Authorized Assembler Services Guide* for detailed guidance on subpool selection.

Notes:

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.

If you omit this parameter, there is no containing boundary.

,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” on page 168 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.

,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.

Notes:

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.

Storage tracking is available as of MVS/SP 4.3. Programs that issue the GETMAIN macro with the OWNER parameter can run on any MVS system from MVS/SP 2.1 to the current release.

,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.

CHECKZERO processing is available as of OS/390 R6. Programs that issue the GETMAIN macro with the CHECKZERO parameter can run on any MVS system from MVS/SP 2.1 to the current release. 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.

Table 23. Return Codes for the GETMAIN Macro (continued)

Return Code	Meaning and Action
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 9. 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=9,OWNER=PRIMARY
```

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.

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, Vol 4 (RD-SRRA)*.

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 “Chapter 4: Serialization” in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

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.

Input Register Information

Before issuing the GQSCAN macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
13	The address of an 18-word save area

© Copyright IBM Corp. 1988, 2002

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

When you specify SCOPE=GLOBAL, or SCOPE=LOCAL, the performance of programs that issue ENQ, DEQ, or the RESERVE macro may be temporarily degraded while the GQSCAN service is running. If you specify XSYS=YES the GQSCAN requestor is suspended until the service completes.

Syntax

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

---

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 or register (2) - (12). *area size*: symbol, decimal digit, or register (2) - (12). **Note:** AREA cannot be specified with QUIT=YES.

,REQLIM=*value* ,REQLIM=MAX	*value*: symbol, decimal digit, register (2) - (12), or the word MAX. **Default**: REQLIM=MAX
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS ,SCOPE=LOCAL ,SCOPE=GLOBAL	**Default**: SCOPE=STEP
,RESERVE=YES ,RESERVE=NO	**Default**: All resources requested with RESERVE and all resources requested with ENQ.
,RESNAME=*(qname addr*[,rname addr, *rname length],* [GENERIC|SPECIFIC], *qname length)*	*qname addr*: RX-type address or register (2) - (12). *rname addr*: RX-type address or register (2) - (12). *rname length*: decimal digit, or register (2) - (12). **Default**: assembled length of rname. **Default**: *qname length* of eight.
,SYSNAME=(*sysname addr [,asid value]*)	*sysname addr*: RX-type address or register (2) - (12). *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 ,QUIT=NO	**Default**: 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 ,XSYS=NO	**Default**: XSYS=YES **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 between 0 and 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.

,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.

,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 between 0 and 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 between 0 and 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 between 0 and 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

None.

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
0	**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.
4	**Meaning**: Queue scan processing is complete. No resources matched your request. **Action**: Do not try to process any data; none exists.
8	**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 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 specifed 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.

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:

---

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.

Notes:

1. This parameter cannot be specified with QUIT=YES.
2. AREA is required on either the list or the execute form of the macro.

,REQLIM=*value* ,REQLIM=MAX	*value*: symbol, decimal digit or the word MAX. **Default**: REQLIM=MAX
,SCOPE=ALL ,SCOPE=STEP ,SCOPE=SYSTEM ,SCOPE=SYSTEMS	**Default**: SCOPE=STEP
,RESERVE=YES ,RESERVE=NO	**Default**: All resources requested with RESERVE and all resources requested with ENQ.
,RESNAME=*(qname addr [,rname addr, rname length], [GENERIC|SPECIFIC], qname length)*	*qname addr*: A-type address. *rname addr*: A-type address. *rname length*: decimal digit. **Default**: assembled length of rname. **Default**: *qname length* of eight.
,SYSNAME=*(sysname addr [,asid value])*	*sysname addr*: A-type address. *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 ,QUIT=NO	**Default**: 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.
,XSYS=YES ,XSYS=NO	**Default**: XSYS=YES **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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede GQSCAN.
GQSCAN	
b	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). **Notes:** 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* ,REQLIM=MAX	*value*: symbol, decimal digit, register (2) - (12), or the word MAX.
,SCOPE=STEP ,SCOPE=ALL ,SCOPE=SYSTEM ,SCOPE=SYSTEMS ,SCOPE=LOCAL ,SCOPE=GLOBAL	**Note:** SCOPE=LOCAL and SCOPE=GLOBAL cannot be coded on the list form of this macro.
,RESERVE=YES ,RESERVE=NO	
,RESNAME=*(qname addr[,rname addr, rname length],* *[GENERIC|SPECIFIC], qname length)*	*qname addr*: RX-type address or register (2) - (12). *rname addr*: RX-type address or register (2) - (12). *rname length*: decimal digit, register (2) - (12). **Default**: assembled length of rname.

,SYSNAME=*(sysname addr [,asid value]*	*sysname addr*: RX-type address or register (2) - (12). *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 ,QUIT=NO	**Default**: 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.
,TOKEN=*addr*	*addr*: RX-type address of register (2) - (12).
,XSYS=YES ,XSYS=NO	**Default**: XSYS=YES **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.

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 MVS Diagnosis: Procedures* for information about using GTF.

Refer to *z/OS MVS Diagnosis: Tools and Service Aids* for descriptions of GTF records.

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 should 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.

As of MVS/ESA SP 4.3.0, issuing GTRACE TEST prior to issuing GTRACE DATA is no longer necessary. 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. Prior to MVS/ESA SP 4.3.0, it was necessary to explicitly issue GTRACE TEST as a separate invocation.

Environment

The requirements for the caller are:

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

© Copyright IBM Corp. 1988, 2002

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

Invokers that do not hold the CPU lock will be informed that user trace records have been requested for any valid EID whenever GTF is active.

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:

---

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	
,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:

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede GTRACE.
GTRACE	
b	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.

Starting with MVS/ESA SP 4.3.0, data is written only if you requested the event qualifier (through the USRP option) when you started GTF. Therefore, if you are running MVS/ESA SP 4.3.0 or later, 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 should 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:

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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede GTRACE.
GTRACE	
b	One or more blanks must follow GTRACE.

DATA=*addr* DATA64=*addr*	*addr*: RX address or register (2) - (12).
,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 ,PAGEIN=YES	**Default**: PAGEIN=NO

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 insure 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.
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.

Table 26. Return Codes for the GTRACE DATA Macro (continued)

Hexadecimal Return Code	Meaning and Action
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:

---

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* DATA64=*addr*	*addr*: A-type address or register (2) - (12).
,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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede GTRACE.
GTRACE	
b	One or more blanks must follow GTRACE.

DATA=*addr* DATA64=*addr*	*addr*: RX address or register (2) - (12).
,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 ,PAGEIN=YES	**Default**: PAGEIN=NO
,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.

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 expanded 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”.
- The ESO hiperspace is backed only with expanded 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 207.

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 the storage range(s) in the address space and the storage range(s) 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:

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

© Copyright IBM Corp. 1988, 2002

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.

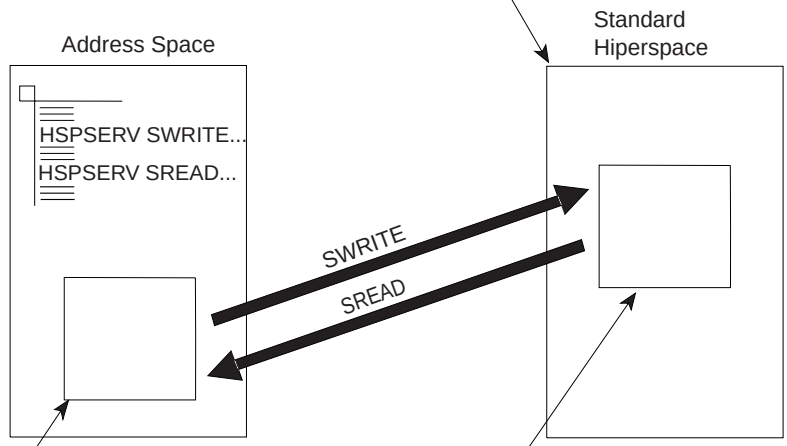
The following figure describes the characteristics and restrictions for the use of standard hiperspaces, the hiperspaces that allow your program to **scroll** through large areas of data.

Non-shared standard hiperspace:

- For problem state and PSW key 8-F callers:
  - If an ALET is not used, the aller's TCB must own the hiperspace.
  - If an ALET is used, any TCB in the caller's home address space can own the hiperspace.
- For superviosr state or PSW key 0-7 callers, any TCB in the caller's home address space can own the hiperspace.
- If an ALET is used:
  - The ALET must be used for a hiperspace on the caller's DU-AL or PASN-AL.
  - The cross memory mode can be any.
- If an ALET is not used, the cross memory mode must be PASN=HASN.
- For PSW key 0 callers, can have any storage key and can be fetch protected.
- For PSW key 1-F callers requesting SWRITE or SREAD RELEASE=YES, must have matching storage key.
- For PSW key 1-F callers requesting SREAD RELEASE=NO, can have non-matching storage key only if hiperspace is not fetch protected.

Shared standard hiperspace:

- Problem state and PSW key 8-F callers must use an ALET.
- Any task in the sytem can own the hiperspace. If the owning task is not in the caller's home or primary address space, the owner's home address space must be non-swappable.
- If an ALET is used, it must be for a hiperspace on the caller's DU-AL or PASN-AL.
- The cross memory mode can be any.
- For PSW key 0 callers, can have any storage key and can be fetch protected.
- For PSW keys 1-F callers requesting SWRITE or SREAD RELEASE=YES, must have matching storage key.
- For PSW key 1-F callers requesting SREAD RELEASE=NO, can have non-matching storage key only if hiperspace is not fetch-protected.



Area of standard hiperspace.

- Must be on 4K boundry.
- Can map a data-in-virtual object on permanent storage.
- For SWRITE requests, cannot have a DIV SAVE current for the area of the hiperspace.
- If an ALET is used, cannot have a DIV SAVE current for any part of the hiperspace.

Area in address space:

- Must be in private subpool.
- Must be within the home address space.
- Must not be within a DREF subpool.
- Can't be page-fixed.
- Must be on a 4K-byte boundary.
- Can't be part of a VIO window.
- For PSW key 0 callers, can have any storage key.
- For PSW key 1-F callers, must have a matching storage key with one exception: for SWRITE callers, if the area is not fetch-protected, it can have any storage key.

Figure 5. Characteristics and Restrictions for Standard Hiperspaces

Syntax

The standard form of the HSPSERV macro for standard hiperspaces is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede HSPSERV.
HSPSERV	
b	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* ,NUMRANGE=*num-addr*	*n*: Number from 1 to 50. *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 ,RELEASE=YES	**Default**: RELEASE=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:

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.

Notes

- 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.
- 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 203.

,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.

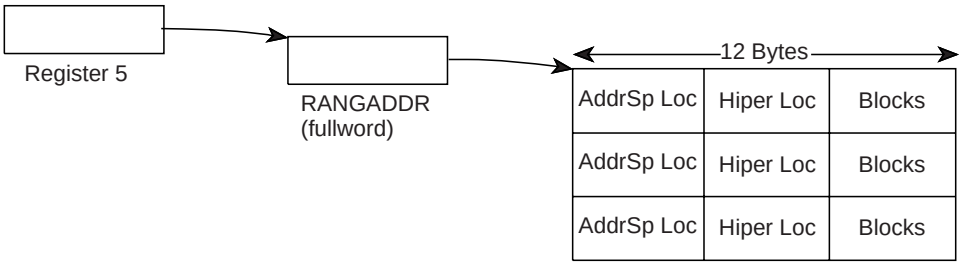
Third word The number of blocks the system is to read or write. Note that the address is the block number followed by 12 binary zeroes.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:

NUMRANGE=3 ,RANGLIST=(5)

or

NUMRANGE=3, RANGLIST=RANGADDR



Further restrictions on the areas in the address space and the hiperspace are described in Figure 5 on page 203.

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 27. 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 due to 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:

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 or disabled reference (DREF) storage. If the caller specifies HSPALET and is disabled, the save area must also be in nonpageable storage. The parameter list, range 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.

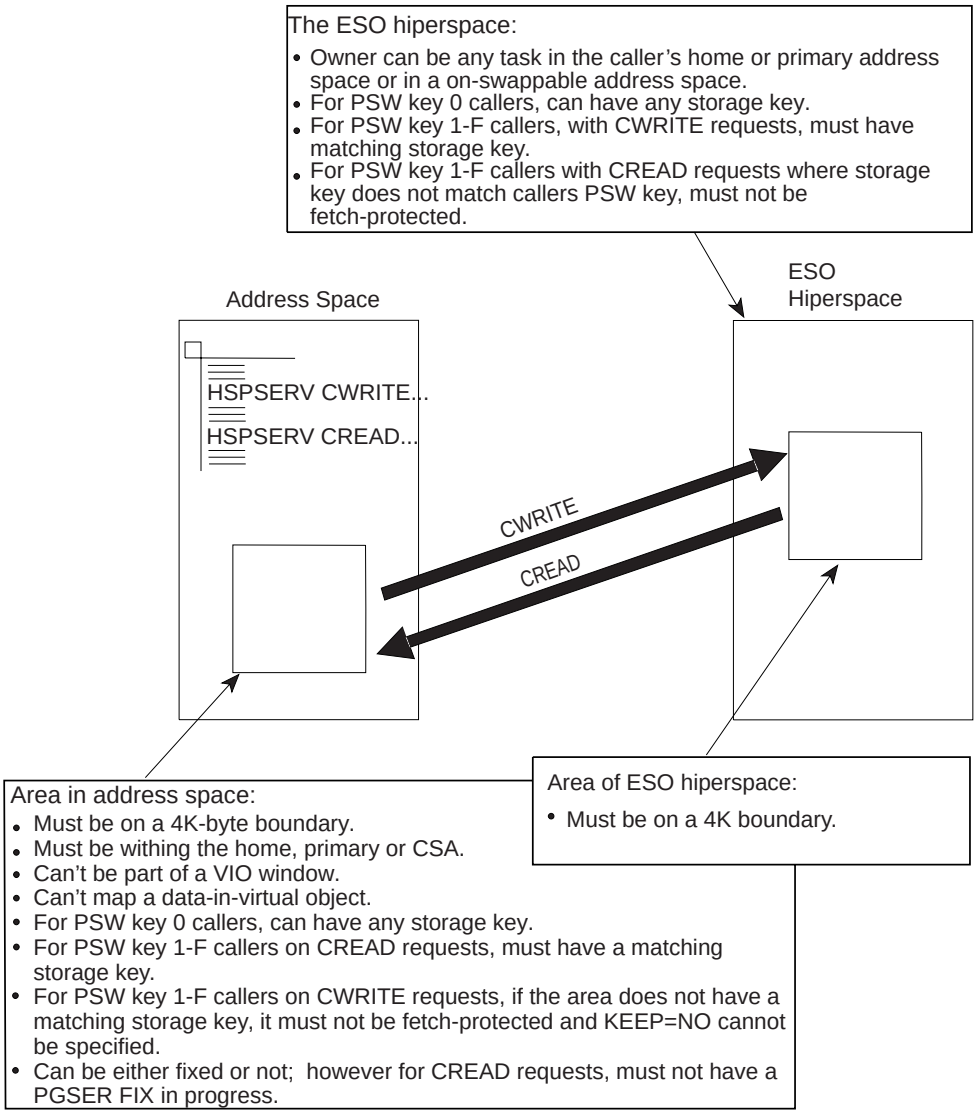


Figure 6. Characteristics and Restrictions for ESO Hiperspaces

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. 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.

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede HSPSERV.

HSPSERV

b	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 ,ADDRSP=PRIMARY ,ADDRSP=COMMON	**Default**: ADDRSP=HOME.
,KEEP=YES ,KEEP=NO	**Default**: KEEP=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:

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 209.

,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.

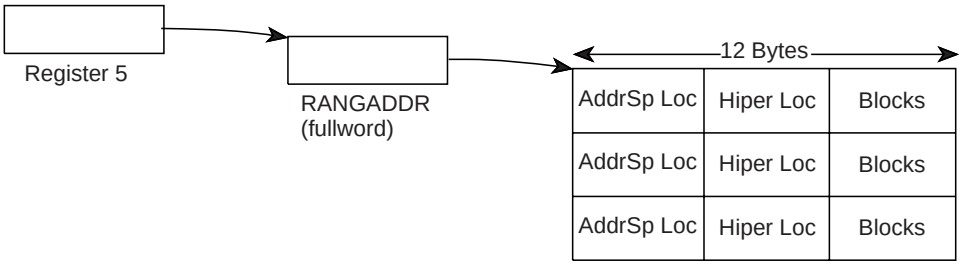
Third word The number of blocks the system is to read or write.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:

NUMRANGE=3 ,RANGLIST=(5)

or

NUMRANGE=3, RANGLIST=RANGADDR



The address space range(s) 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 209.

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 28. 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 central (also called real) or expanded storage. You can repeat the HSPSERV request after you reference the range(s), 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 central storage frames or expanded 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.

Table 28. Return and Reason Codes for HSPSERV CREAD and HSPSERV CWRITE (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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:

---

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)* ,MF=(L,*list addr,attr)*	*list-addr*: Symbol. *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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede HSPSERV.
HSPSERV	
b	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 ,NUMRANGE=*num-addr*	**Default**: NUMRANGE=1. *num-addr*: RX-type address or register (2) - (12).
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).
,RELEASE=NO ,RELEASE=YES	**Default**: RELEASE=NO.
,ADDRSP=HOME ,ADDRSP=PRIMARY ,ADDRSP=COMMON	**Default**: ADDRSP=HOME.
,KEEP=YES ,KEEP=NO	**Default**: KEEP=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=(E,*list-addr,*COMPLETE) ,MF=(E,*list-addr*,NOCHECK)	*list-addr*: RX-type address or register (2) - (12). **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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede HSPSERV.
HSPSERV	
b	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 ,NUMRANGE=*num-addr*	**Default**: NUMRANGE=1. *num-addr*: RX-type address or register (2) - (12).
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).
,RELEASE=NO ,RELEASE=YES	**Default**: RELEASE=NO.
,ADDRSP=HOME ,ADDRSP=PRIMARY ,ADDRSP=COMMON	**Default**: ADDRSP=HOME.
,KEEP=YES ,KEEP=NO	**Default**: KEEP=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=(E,*list-addr*,COMPLETE) ,MF=(E,*list-addr*,NOCHECK)	*list-addr*: RX-type address or register (2) - (12). **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.

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:

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 may hold the local or CPU lock, but is not required to hold any locks.
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

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IARR2V.
IARR2V	
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* ,WORKREG=NONE	*work_reg*: RS-type address, or register (2) - (12). **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 ,LINKAGE=BRANCH	**Default**: LINKAGE=SYSTEM
,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 double-word 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 fullword 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 29. 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.

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
```

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 on 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:

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 on page 228 for the required attributes of this storage. The caller must not release this storage until after issuing IARSUBSP UNIDENTIFY.

Restrictions

None.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IARSUBSP.
IARSUBSP	
b	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 ,GENNAME=COND ,GENNAME=YES	**Default**: GENNAME=NO
,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. Once 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 30. 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 31. 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 31. 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 due to 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 due to 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 due to a shortage of resources. **Action**: Reissue the request. If the problem persists, contact your system programmer.

Table 31. Return and Reason Codes for the IARSUBSP Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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:

---

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*) MF=(L,*list addr,attr*) MF=(L,*list addr*,0D)	*list addr*: symbol. *attr*: 1- to 60-character input string. **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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IARSUBSP.
IARSUBSP	
b	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 ,GENNAME=COND ,GENNAME=YES	**Default**: GENNAME=NO
,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*) ,MF=(E,*list addr*,COMPLETE) ,MF=(E,*list addr*,NOCHECK)	*list addr*: RX-type address or address in register (2) - (12). **Default:** COMPLETE

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.

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:

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. For information on the IARVRL macro, see *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

© Copyright IBM Corp. 1988, 2002

- 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 32. 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 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.

Syntax

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

---

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 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.

,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 33. 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 33. 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 33. 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IARVSERV.
IARVSERV	
b	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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IARVSERV.
IARVSERV	
b	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 ,RETAIN=YES	**Default**: RETAIN=NO
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default**: IMPLIED_VERSION *plistver*: 0
,MF=(E,*list addr*) ,MF=(E,*list addr*,COMPLETE) ,MF=(E,*list addr*,NOCHECK)	*list addr*: RX-type address or address in register (2) - (12). **Default**: COMPLETE

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.

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 or 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. The area above the bar is intended to be used for data only, not for executing programs. 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 memory object you can specify a guard area (not accessible) and a usable area. Later, you can change all or some of a guard area into an accessible area and vice versa.

The following services are provided:

GETSTOR
: Create a memory object (Page 248)

PAGEFIX
: Fix physical pages within one or more memory objects. (Page 255)

PAGEUNFIX
: Unfix physical pages within one or more memory objects. (Page 260)

PAGEOUT
: Notify the system that data within physical pages of one or more memory objects will not be used in the near future. (Page 264)

PAGEIN
: Notify the system that data within physical pages of one or more memory objects are needed in the near future. (Page 269)

DISCARDDATA
: Discard data within physical pages of one or more memory objects. (Page 273)

CHANGEGUARD
: Request that a specified range in a memory object be changed from guard area to usable area or vice versa. (Page 277)

LIST Requests a list of memory objects. (Page 282)

DETACH
: Free one or more memory objects. (Page 286)

For guidance information on the use of 64–bit virtual storage allocation, see *z/OS MVS Programming: Extended Addressability Guide*.

© Copyright IBM Corp. 1988, 2002

After the separate descriptions of each individual Request are the following sections which apply to all of the Requests:
- The abend codes on page 292,
- The return and reason codes on page 293, and
- Examples of using IARV64 on page 293

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 one megabyte boundary.
- The limit of the amount 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 category of storage does not count against the MEMLIMIT:
  – The guard area in a memory object.

REQUEST=GETSTOR Option of IARV64

REQUEST=GETSTOR allows you to create a memory object. To avoid an abend for exceeding MEMLIMIT, specify the COND=YES parameter.

Environment

The requirements for the caller are:

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: • KEY • CONTROL=AUTH • ALETVALUE
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN **Note:** that problem state caller running in PSW key 8-15 can use GETSTOR/DETACH only when primary = home.
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, subject to the following limitation: 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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IARV64.
IARV64	
b	One or more blanks must follow IARV64.

REQUEST=GETSTOR	
,COND=**NO** ,COND=YES	**Default:** COND=NO
,SEGMENTS=*segments*	*segments:* RS-type address or address in register (2) - (12).

,KEY=*key*	*key:* RS-type address or address in register (2) - (12).
,KEY=**CALLERKEY**	**Default:** KEY=CALLERKEY
,FPROT=**YES**	**Default:** FPROT=YES
,FPROT=NO	
,SVCDUMPRGN=**YES**	**Default:** SVCDUMPRGN=YES
,SVCDUMPRGN=NO	
,CONTROL=**UNAUTH**	**Default:** CONTROL=UNAUTH
,CONTROL=AUTH	
,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
,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).
,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 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 memory object. At completion, the memory object is created in the address space you indicate on the ALETVALUE parameter. If you don't specify ALETVALUE, the system creates the memory object in the primary address space. Note that problem state routines running in PSW key 8-15 can use GETSTOR only when primary = home. When the memory object owner terminates, the memory object will be freed.

,COND=NO
,COND=YES

An optional parameter that specifies whether the request is unconditional or conditional. In all cases the request will be ABENDed for invalid requests, including violation of environmental restrictions. The default is COND=NO.

,COND=NO

The request is unconditional. The request will be ABENDed when the request cannot be satisfied.

,COND=YES

The request is conditional. The request will not be ABENDed for resource unavailability.

,SEGMENTS=*segments*

A required input parameter that specifies the size of the memory object requested in megabytes. This must be a non-zero 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.

,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 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.

,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
An SVC dump will include in its virtual storage capture for the owning address space the usable area of the memory object whenever SDATA=RGN is specified.

,SVCDUMPRGN=NO
The SVC dump option SDATA=RGN will not include the virtual storage of this memory object in the dump.

,CONTROL=UNAUTH
,CONTROL=AUTH
An optional 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.

,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 high-order half (bits 0-31) of the user token must be binary zeros for a problem state program with PSW key 8 - 15. The system enforces this requirement. 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.

The convention for supervisor state program with PSW key 0-7, is that the high-order half (bits 0-31) should represent an address of some storage related to the caller. The system enforces the rule that the high-order half is non-zero for authorized callers. The format for the right word is a choice left to the authorized 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
An optional 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.

,GUARDLOC=LOW
,GUARDLOC=HIGH
An optional 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 ABENDed 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.

To code: Specify the RS-type address, or address in register (2)-(12), of an eight-byte 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=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.

REQUEST=PAGEFIX Option of IARV64

REQUEST=PAGEFIX allows you to fix physical pages within one or more memory objects. It makes virtual storage areas, above the bar, resident in central (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:

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 above and below the bar.

Programming Requirements

None

Restrictions

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

---

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=PAGEFIX

,LONG=**YES** ,LONG=NO	**Default:** LONG=YES
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).

,ALETVALUE=*aletvalue* ,ALETVALUE=**0**	*aletvalue:* RS-type address or address in register (2) - (. **Default:** ALETVALUE=0
,NUMRANGE=*numrange* ,NUMRANGE=**1**	*numrange:* RS-type address or address in register (2) - (12). **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** ,PLISTVER=MAX ,PLISTVER=0	**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**)	**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=PAGEFIX
A required parameter. REQUEST=PAGEFIX specifies that the data within the specified ranges be pagefixed.

PAGEFIX can not be requested for a memory object that was created using GETSTOR CONTROL=UNAUTH.

PAGEFIX can not 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 cannot be freed; if the system finds a PAGEFIXed area in the memory object, it abends the DETACH caller.

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 prepartation 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 will be 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 via 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 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
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 page boundary.

The length of this field is 8 bytes.

NUMPAGES
contains the number of physical pages in the area.

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 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 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

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.

REQUEST=PAGEUNFIX Option of IARV64

REQUEST=PAGEUNFIX allows you to unfix physical pages within one or more memory objects.

Environment

The requirements for the caller are:

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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IARV64.
IARV64	
b	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
,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 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.

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 ABENDed.

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 page boundary.

The length of this field is 8 bytes.

NUMPAGES
contains the number of physical pages in the area.

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 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 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.

,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
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.

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:

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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IARV64.
IARV64	
b	One or more blanks must follow IARV64.

REQUEST=PAGEOUT	
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue* ,ALETVALUE=**0**	*aletvalue:* RS-type address or address in register (2) - (12). **Default:** ALETVALUE=0
,NUMRANGE=*numrange* ,NUMRANGE=**1**	*numrange:* RS-type address or address in register (2) - (12). **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** ,PLISTVER=MAX ,PLISTVER=0	**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**)	**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=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 returned by GETSTOR CONTROL=AUTH.

The value must always be on a physical page boundary.

The length of this field is 8 bytes.

NUMPAGES
contains the number of physical 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 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

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.

REQUEST=PAGEIN Option of IARV64

REQUEST=PAGEIN notifies the system that data within physical pages of one or more memory objects are needed in the near future.

Environment

The requirements for the caller are:

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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IARV64.
IARV64	
b	One or more blanks must follow IARV64.

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	
,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.

It must always be on a physical page boundary.

The length of this field is 8 bytes.

NUMPAGES
contains the number of physical 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 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 allocated.

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
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.

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:

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:	A local lock may be held, subject to the following limitation: 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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

---

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=DISCARDDATA	
,CLEAR=**YES** ,CLEAR=NO	**Default:** CLEAR=YES
,RANGLIST=*ranglist*	*ranglist:* RS-type address or address in register (2) - (12).
,ALETVALUE=*aletvalue* ,ALETVALUE=**0**	*aletvalue:* RS-type address or address in register (2) - (12). **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** ,PLISTVER=MAX ,PLISTVER=0	**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**)	**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=DISCARDDATA
A required parameter. REQUEST=DISCARDDATA discards the data within the specified ranges.

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 area or guard area in the area to be discarded, the caller will be ABENDed. Any prior pages processed will have data in an indeterminate state when CLEAR=NO was used.

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.

,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.

The value must always be on a physical page boundary.

The length of this field is 8 bytes.

NUMPAGES
contains the number of physical 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 an eight-byte pointer field.

,ALETVALUE=*aletvalue*
,ALETVALUE=0
An optional input parameter that indicates the ALET of the 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.

,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
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.

REQUEST=CHANGEGUARD Option of IARV64

REQUEST=CHANGEGUARD requests that a specified amount of a memory object be changed from the guard area to the usable area or vice versa. To avoid an abend for exceeding the MEMLIMIT, specify the COND=YES parameter.

Environment

The requirements for the caller are:

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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IARV64.
IARV64	
b	One or more blanks must follow IARV64.

REQUEST=CHANGEGUARD	
,CONVERT=TOGUARD ,CONVERT=FROMGUARD	
,COND=**NO** ,COND=YES	**Default:** COND=NO
,ALETVALUE=*aletvalue* ,ALETVALUE=**0**	*aletvalue:* RS-type address or address in register (2) - (12). **Default:** ALETVALUE=0
,MEMOBJSTART=*memobjstart*	*memobjstart:* RS-type address or address in register (2) - (12).
,CONVERTSIZE=*convertsize*	*convertsize:* 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** ,PLISTVER=MAX ,PLISTVER=0	**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**)	**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=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 ABENDed.

When COND=YES and there is insufficient storage to satisfy the request, instead of the request being ABENDed, the request will complete, but a return code will be set to indicate that the request could not be completed successfully.

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 caller must be the owner of the memory object.

,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 guard areas. The data in the converted area will be released. This operation reduces the amount of virtual storage that contributes toward the MEMLIMIT for the address space identified by ALETVALUE.

When GUARDLOC=LOW is specified, the first usable virtual address in the memory object is increased.

When GUARDLOC=HIGH is specified, the last usable virtual address in the memory object is decreased.

,CONVERT=FROMGUARD

Convert the specified amount of guard area to be usable area. The converted (now usable) area will appear as pages of zeros. This operation increases the amount of area that contributes toward the MEMLIMIT for the address space designated by ALETVALUE.

When GUARDLOC=LOW is specified, the first usable virtual address in the memory object is decreased.

When GUARDLOC=HIGH is specified, the last usable virtual address in the memory object is increased.

,COND=NO
,COND=YES

An optional parameter that specifies an unconditional or conditional request. In all cases the request will be ABENDed for invalid requests, including violation of environmental restrictions. The default is COND=NO.

,COND=NO

The request is unconditional. The request will be ABENDed when the request cannot be satisfied.

,COND=YES

The request is conditional. The request will not be ABENDed when a MEMLIMIT violation would occur.

,ALETVALUE=*aletvalue*
,ALETVALUE=0

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.

,MEMOBJSTART=*memobjstart*
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 eight-byte pointer field.

,CONVERTSIZE=*convertsize*
A required input parameter, that indicates the number of 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, CONVERTSIZE 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 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.

,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
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.

REQUEST=LIST Option of IARV64

REQUEST=LIST requests a list of objects be provided to the caller.

Environment

The requirements for the caller are:

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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IARV64.
IARV64	

b	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).
,V64SELECT=**NO** ,V64SELECT=YES	**Default:** V64SELECT=NO
,SVCDUMPRGN=**YES** ,SVCDUMPRGN=NO	**Default:** SVCDUMPRGN=YES
,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	**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**)	**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=LIST
A required parameter. REQUEST=LIST provides information about the usable areas (not guard areas) of memory objects.

The information is returned in a work area that you specify, in a format described in IARV64WA. Information includes starting and ending address and storage key.

,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.

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 64 bytes long.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,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

,V64SELECT=YES
The request is for a subset of the allocated memory objects. Only memory object which meet the selection criteria will be returned. If no selection criteria is provided or no memory object meets the selection criteria then no object will be returned.

,SVCDUMPRGN=YES
,SVCDUMPRGN=NO
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.

,SVCDUMPRGN=YES
The memory object will be included in the set of memory objects returned.

,SVCDUMPRGN=NO
The memory object will not be included in the set of memory objects returned.

,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
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.

REQUEST=DETACH Option of IARV64

REQUEST=DETACH allows you to free one or more memory objects.

Environment

The requirements for the caller are:

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 OWNER=NO parameter 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:** Note that problem state caller running in PSW key 8-15 can use GETSTOR/DETACH only when primary = home.
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, subject to the following limitation: 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

No subspace can be defined for the address space specified by the ALETVALUE, except for the REQUEST=LIST request.

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=DETACH option of the IARV64 macro is written as follows:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IARV64.
IARV64	
b	One or more blanks must follow IARV64.

REQUEST=DETACH

,MATCH=**SINGLE** ,MATCH=USERTOKEN	**Default:** MATCH=SINGLE
,MEMOBJSTART=*memobjstart*	*memobjstart:* 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
,USERTKN=*usertkn*	*usertkn:* RS-type address or address in register (2) - (12).
,OWNER=**YES** ,OWNER=NO	**Default:** OWNER=YES
,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
,COND=**NO** ,COND=YES	**Default:** COND=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

,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**)

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=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 primary = home, 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.

A memory object can be affected by DETACH when MATCH=SINGLE USERTKN=NO_USERTKN is specified, even when the memory object has an associated user token. Other invocations of DETACH will affect memory objects only when a matching user token is passed.

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, but any prior pages will have indeterminate data and the caller will be ABENDed.

,MATCH=SINGLE
,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=USERTOKEN
Specifies that the input contains a user token that was passed to GETSTOR. Memory objects not associated with a user token are not affected. Such objects would have to have been created using GETSTOR NO_USERTKN. If MATCH=USERTOKEN, COND=YES, and no matching user token exists, the system returns a return code instead of abending the caller. All memory objects associated with this user 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.

If the system encounters an error in processing a qualifying memory object (e.g. unexpected pagefixed page), the processing ends. The system does not process that page or any further pages or memory objects and abends the caller.

,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 eight-byte pointer field.

,USERTKN=*usertkn*
,USERTKN=NO_USERTKN
When MATCH=SINGLE is specified, an optional input parameter that identifies the user token to uniquely identify the memory object, as previously passed to GETSTOR.

When the memory object is not associated with the input token value, it will not be processed. The default is NO_USERTKN.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,USERTKN=*usertkn*
When MATCH=USERTOKEN is specified, a required input parameter that identifies the user token.

To code: Specify the RS-type address, or address in register (2)-(12), of a doubleword field.

,OWNER=YES
,OWNER=NO
An optional parameter 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. 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 is 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. The mother task may not be given however 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 ABENDed 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.

,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.

,COND=NO
,COND=YES
An optional parameter that specifies whether the request is unconditional or conditional. In all cases the request will be abended for invalid requests, including violation of environmental restrictions. The default is COND=NO.

,COND=NO
The request is unconditional. The request will be abended when the request cannot be satisfied.

,COND=YES
The request is conditional. A REQUEST=DETACH, MATCH=USERTOKEN which does not select any memory objects will not be abended.

,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
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

IARV64 might abnormally terminate with hexidecimal abend code DC2. See *z/OS MVS System Codes* for an explanation and programmer response.

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.

The following table identifies the hexadecimal return and reason codes. IBM support personnel may request the entire reason code, including the **xx** value.

Table 34. 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**: Re-issue the call if you need the information pertaining to those recent IARV64 requests.
04	—	**Meaning**: Successful completion, with exception. For a LIST request, that there are additional MOMBs which were not returned on this calls to LIST. **Action**: Issue the LIST call again to get the additional information.
06	—	**Meaning**: Successful completion, with exception. For a LIST request, there are additional MOMBs which were not returned on this call to LIST and IARV64 requests have been issued since the previous call to LIST. **Action**: Issue the LIST call again to get the additional information.
08	—	Meaning; The request is rejected due to non-system failure. For a GETSTOR or CHANGEGUARD request, there was insufficient storage to satisfy the request. For a DETACH request, there were no MOMBs deleted because none match the user token provided. For a LIST request, there were no MOMBs returned because no MOMBs match the selection criteria. **Action**: For a GETSTOR or CHANGEGUARD request, re-issue the call with a smaller storage size. For a DETACH request, make sure that the user token was correct. For a LIST request, no action is required.

Example

Operation:

1. Get 2M above the bar
2. Page-fix the first 1M of that storage
3. Page-unfix that first 1M
4. Free the storage

The code is as follows:

```
         SYSSTATE AMODE64=YES
*************************************************************
* Get storage above 2G                                      *
*************************************************************
         IARV64 REQUEST=GETSTOR,SEGMENTS=NUMSEG,            *
               ORIGIN=O,                                    *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,V64L)
*
* Place code to check return/reason codes here
*
*
* Build the Range List for Pagefix
*
         LG    1,O
         STG   1,RLSTART
         LG    1,ONEMEG         Number of pages in 1-meg
         STG   1,RLPAGES
         LA    1,RL             Must be in 64bit amode
         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=O,               *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,V64L)
*
* Place code to check return/reason codes here
*
NUMSEG   DC    D'2'
ONEMEG   DC    AD(256)
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
O        DS    AD
RLADDR   DS    AD
RL       DS    2AD
         ORG   RL
RLSTART  DS    AD
RLPAGES  DS    AD
         IARV64 MF=(L,V64L)
```

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

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 IAZCXTKN 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:

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).

© Copyright IBM Corp. 1988, 2002

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 35. 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
```

IAZXJSAB — Obtain Information about a Currently Running Job

Description

Use the IAZXJSAB macro to obtain information about the job currently running in an address space. Depending on the parameters you specify for the IAZXJSAB macro, the system returns information such as:

- The name of the subsystem that scheduled the job (COMPID parameter)
- The job identifier (WORKID or JOBID parameter)
- The job name (JOBNAME parameter)
- The prefix of messages associated with the job (PREFIX parameter)
- The user ID associated with the job (USERID parameter)
- The time when the job started running (EXECST parameter)
- The XCF group name of the subsystem (XCFGPNM parameter)
- The JES status of the job (JESTAT parameter)
- The level of the JSAB used for the READ request (JSABLVL parameter).

Environment

The requirements for the caller are:

Minimum authorization: Problem 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, Vol 3 (IVT-RCWK)*	JSAB
IHAASCB	*z/OS MVS Data Areas, Vol 1 (ABEP-DALT)*	ASCB
IHAASSB	*z/OS MVS Data Areas, Vol 1 (ABEP-DALT)*	ASSB
IHAPSA	*z/OS MVS Data Areas, Vol 3 (IVT-RCWK)*	PSA
IHASTCB	*z/OS MVS Data Areas, Vol 5 (SSAG-XTLST)*	STCB
IKJTCB	*z/OS MVS Data Areas, Vol 5 (SSAG-XTLST)*	TCB

Restrictions

None.

© Copyright IBM Corp. 1988, 2002

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:

---

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.

READ

,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).
,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).
,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).

Parameters

The parameters are explained as follows:

READ
Requests that the system return information about the job currently running in the address space.

,ASCB=*ascb addr*
Specifies the address of an address space control block (ASCB). The system obtains information about the job from the job scheduler address space control block (JSAB) associated with this ASCB. The default value is the address of the ASCB that represents the caller's home address space.

,COMPID=*compid addr*
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 JES subsystem name. If APPC/MVS created the JSAB, the identifier is ASCH.

,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.

,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.

,JOBNAME=*jobname addr*
Specifies the location where the system is to return the 8-character job name.

,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.

,USERID=*userid addr*
Specifies the location where the system is to return the 8-character user ID.

,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.

,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.

,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, Vol 3 (IVT-RCWK)*.

,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 0, it indicates an address space level JSAB; a 4 indicates a subtask level JSAB.

ABEND Codes

None.

Return Codes

When IAZXJSAB macro returns control to your program, GPR 15 contains one of the following hexadecimal return codes.

Table 36. Return and Reason Codes for the IAZXJSAB Macro

Return Code	Meaning and Action
0	**Meaning**: Processing completed successfully. **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.

Example

Obtain the job id of the current address space.

```
         IAZXJSAB READ,JOBID=MYJOBID
          .
          .
          .
MYJOBID  DS    CL8
```

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.

Environment

The requirements for the caller are:

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
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.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IEAARR.
IEAARR	
b	One or more blanks must follow IEAARR.

ARRPTR=*arrptr*	*arrptr:* RX-type address or address in register (2) - (12).
,ARRPARAMPTR=*arrparamptr*	*arrparamptr:* RX-type address or address in register (2) - (12).
,PARAMPTR=*paramptr*	*paramptr:* RX-type address or address in register (2) - (12).
,TARGETPTR=*targetptr*	*targetptr:* RX-type address or address in register (2) - (12).
,TARGETSTATE=PROB ,TARGETSTATE=SUP	

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.

ARRPTR=*arrptr*
A required input parameter that contains the address of the associated recovery routine. This routine gets control from RTM according to normal OS/390 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.

,ARRPARAMPTR=*arrparamptr*
A required input parameter that contains the address of the parameter area to be passed to the ARR upon error. The address is placed in the first four bytes of the area pointed to by SDWAPARM. Note that the second four bytes of that area will not contain interface information.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.

,PARAMPTR=*paramptr*
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.

,TARGETPTR=*targetptr*
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.

To code: Specify the RX-type address, or address in register (2)-(12), of a pointer field.

,TARGETSTATE=PROB
,TARGETSTATE=SUP
A required parameter that indicates the requested PSW state of the target routine.

,TARGETSTATE=PROB
indicates 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 the target routine is to get control in supervisor state. This should only be used by a caller currently in supervisor state.

ABEND Codes

The caller may get the following abend code:

0C2-02 TARGETSTATE=SUP was requested by a caller currently running in problems tate.

Return Codes

None.

Example 1

Operation:

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
   ...
```

IEAFP — Floating Point Services

Description

IEAFP allows you to request that the system stop its status saving of the additional floating point status provided with this release. This status consists of the additional floating point (AFP) registers - FPRs 1,3,5,7-15 and the floating point control (FPC) register.

You would typically use this service only when you are a server task which ″subdispatches″ unrelated units of work (e.g., 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.

Environment

The requirements for the caller are:

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. The caller may hold the local, CMS, or CPU lock.
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:

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IEAFP.
IEAFP	
b	One or more blanks must follow IEAFP.

STOP	
,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:

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.

STOP
A required input parameter, keyword that indicates to stop saving additional floating point status until such time as a new floating point operation requires it.

,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).

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 37. 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 parmeters that are not valid. **Action**: Refer to the action provided with the specific reason code.
8	xxxx0801	**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	xxxx0C01	**Equate Symbol**: IeafpRsnFromAsynchExit **Meaning**: IEAFP was issued from an asynchronous exit routine. **Action**: Avoid issuing IEAFP from an asynchronous exit routine.

Example

Operation:

1. Stop additional status saving

The code is as follows.

IEAFP STOP

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 section 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:

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

© Copyright IBM Corp. 1988, 2002

ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled
Interrupt status:	Enabled or disabled for I/O and external interrupts
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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEALSQRY.

IEALSQRY

b	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 38. 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.
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.

Table 38. Return Codes for IEALSQRY (continued)

Return Code	Meaning and Action
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
```

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:

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.

© Copyright IBM Corp. 1988, 2002

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 ASVT and CVT in *z/OS MVS Data Areas, Vol 1 (ABEP-DALT)*, DSD in *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*, and PSA in *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)*.

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 page-fixed.

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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IEAMRMF3.
IEAMRMF3	
b	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 page-fixed.

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 39. 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 should 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
```

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) may 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:

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.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEAMSCHD macro.
IEAMSCHD	
ƀ	One or more blanks must follow IEAMSCHD macro.

EPADDR=*epaddr*	*epaddr*: RS-type address or register (2) - (12).
,ENV=HOME ,ENV=PRIMARY ,ENV=FULLXM ,ENV=STOKEN	**Default**: HOME
,TARGETSTOKEN=*targetstoken*	*targetstoken*: RS-type address or register (2) - (12).
,FEATURE=NONE	**Default**: NONE

,FEATURE=CRYPTO ,FEATURE=CPMASK	
,CPUMASK=*cpumask*	*cpumask*: RS-type address or register (2) - (12).
,PRIORITY=LOCAL ,PRIORITY=GLOBAL ,PRIORITY=CURRENT	**Default**: LOCAL
,PRIORITY=PREEMPT ,PRIORITY=CLIENT ,PRIORITY=ENCLAVE	
,MINORPRIORITY=ZERO	**Default**: ZERO
,MINORPRIORITY=*minorpriority*	*minorpriority*: RS-type address or register (2) - (12).
,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).
,KEYVALUE=INVOKERKEY	**Default**: INVOKERKEY
,KEYVALUE=*keyvalue*	*keyvalue*: RS-type address or register (2) - (12).
,LLOCK=NO ,LLOCK=YES	**Default**: NO
,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).
,SYNCH=NO ,SYNCH=YES	**Default**: NO
,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).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX	**Default**: PLISTVER=IMPLIED_VERSION

```
,PLISTVER=0
,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:

EPADDR=*epaddr*
Specifies the address of the SRB routine to be scheduled for asynchronous execution.

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).

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=ENCLAVE, and PRIORITY=CLIENT.

,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=*enclavtoken* is required for PRIORITY=ENCLAVE.

,FEATURE=NONE
,FEATURE=CRYPTO
,FEATURE=CPMASK
Optional parameter that specifies affinity to specific processors.

NONE
Specifies that there is no affinity to specific processors.

Default: NONE

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.

CPMASK
Specifies that the SRB routine is to be given affinity to only those processors indicated in the CPUMASK=*cpumask* parameter.

,CPUMASK=*cpumask*
Specifies the CP mask for the processor to which you want affinity. Bits that have a value of 1 (on) indicate that the SRB routine is to have affinity to the processor with the same physical address as the bit position. For example, if the fourth (4th) bit is on in the mask, the SRB routine will have affinity to the processor with physical address 3. To schedule an SRB routine with affinity to CP 0, CP 3, and CP 4, the bit mask would be B'1001 1000 0000 0000'.

To code: Specify the name (RS-type), or address in register (2)-(12), of a 16-bit input parameter. CPUMASK=*cpumask* is required for FEATURE=CPMASK.

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 address of the functional recovery routine (FRR) that is to be established prior to the SRB routine receiving control.

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

To code: Specify the name (RS-type), or address in register (2)-(12), of an optional 4-byte input parameter.

,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). The scheduling program can specify an RMTR to be invoked through the PURGEDQ macro. The RMTR is responsible for cleaning up resources on behalf of an SRB routine if it is to be purged before it is dispatched.

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 an 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.

,SYNCH=NO
,SYNCH=YES
Specifies whether or not 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.

Default: NO

,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.

,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).

To code: Specify the RS-type address of a fullword field, or register (2)-(12).

,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. 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 only the following parameters:

CLIENTSTOKEN	FRRADDR	PRIORITY
CPUMASK	KEYVALUE	PTCBADDR
ENCLAVETOKEN	LLOCK	PURGESTOKEN
ENV	MINORPRIORITY	RMTRADDR

EPADDR	PARM	TARGETSTOKEN
FEATURE	PLISTVER	

- **1**, if you use any of the following parameters, and parameters from version 0, or both:

SYNCH	SYNCHCOMPADDR
SYNCHCODEADDR	SYNCHRSNADDR

To code: Specify one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0 or 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

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 40. 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. The values returned on SYNCHCOMPADDR, SYNCHCODEADDR, and SYNCHRSNADDR contain additional information.

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
```

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
```

IEANTCR — Create a Name/Token Pair

Description

Call the IEANTCR service to create a name/token pair.

Environment

The requirements for the caller are:

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
*
* 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.

© Copyright IBM Corp. 1988, 2002

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.

```
                          |
                          | ,(level
                          | ,user_name
CALL  IEANTCR             | ,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:
- 0 - system-level pair will not persist
- 1 - system-level pair will persist.

Note: Only system-level name/token pairs can persist after the creating task terminates. 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 should take:

Table 41. 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*.
24	**Meaning**: The caller specified an incorrect *persist_option*. **Action**: Respecify zero or one 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
```

IEANTDL — Delete a Name/Token Pair

Description

Call the IEANTDL service to delete a name/token pair.

Environment

The requirements for the caller are:

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
```

© Copyright IBM Corp. 1988, 2002

Restrictions

Do not use the IEANTDL callable service in a RESMGR resource manager routine.

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.

```
                          |
                          | ,(level
CALL  IEANTDL             | ,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 should take.

Table 42. 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.

Table 42. Return Codes for the IEANTDL Macro (continued)

Hexadecimal Return Code	Meaning and Action
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.

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:

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
```

© Copyright IBM Corp. 1988, 2002

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 348 for the list of definitions IFBNTASM provides.

Restrictions

- Do not use the IEANTRT callable service in a RESMGR resource manager routine.
- 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.

```
                    |
                    |  ,(level
CALL  IEANTRT       |  ,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 should take.

Table 43. 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 should 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
```

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:

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.

© Copyright IBM Corp. 1988, 2002

Performance Implications

None.

Syntax

The IEARBUP macro is written as follows:

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IEARBUP. IEARBUP
b	One or more blanks must follow IEARBUP.

WHICHRB=CURRENT WHICHRB=PREV	
,PSWBYTE03=NO ,PSWBYTE03=YES	
,ADDRTYPE=**NO_CHANGE** ,ADDRTYPE=INRBOPSWA ,ADDRTYPE=ACTUAL ,ADDRTYPE=DELTA	**Default:** ADDRTYPE=NO_CHANGE
,PSWADDR=*pswaddr*	*pswaddr:* RS-type address or address in register (2) - (12)
,AMODE=UNCHANGED ,AMODE=24 ,AMODE=31 ,AMODE=64	**Default:** AMODE=UNCHANGED
,PSWDELTA=*pswdelta*	*pswdelta:* 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** ,PLISTVER=MAX ,PLISTVER=0	**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**)	**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 IEARBUP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

WHICHRB=CURRENT
WHICHRB=PREV
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.

,PSWBYTE03=NO
,PSWBYTE03=YES
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=NO_CHANGE
,ADDRTYPE=INRBOPSWA
,ADDRTYPE=ACTUAL
,ADDRTYPE=DELTA
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 AMODE indicator(s).

,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.

,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.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-byte field.

,AMODE=UNCHANGED
,AMODE=24
,AMODE=31

,AMODE=64
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.

,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.

,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
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-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.

,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.

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 44. 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.
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
```

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. A transaction dump can request that the dump be written to a data set that is either pre- or automatically allocated. A pre-allocated data set is provided by specifying an open MACRF=W DCB, which must contain sufficient space in one or more extents for the entire dump to be written. Failing to provide a sufficiently large data set will result in a partial dump. Automatic allocation is requested by specifying a data set name pattern, similar to the pattern used for the operator DUMPDS NAME=parameter. Automatic allocation ensures that a dump will not be truncated due to space contraints, and is done using the generic allocation unit name of SYSALLDA. When a dump is written, message IEA822I is 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.

An asynchronous dump may be requested by specifying ASYNC=YES on the dump request. It is recommended that an ECB be specified for asynchronous dumps to ensure that any volatile resources are captured before being freed.

© Copyright IBM Corp. 1988, 2002

Environment

The requirements for the caller are:

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
Locks:	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 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

If the caller is passing a DCB to transaction dumping, it must be opened before the invocation.

Restrictions

The caller may not have any FRRs established.

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 IEATDUMP macro is written as follows:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEATDUMP.
IEATDUMP	
b	One or more blanks must follow IEATDUMP.

DCBAD=*dcbad*	*dcbad*: RS-type address or register (2) - (12).
DCB=*dcb*	*dcb*: RS-type address or register (2) - (12).
DSNAD=*dsnad*	*dsnad*: RS-type address or register (2) - (12).
DSN=*dsn*	*dsn*: RS-type address or register (2) - (12).
,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).
,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** ,SDATA=ALLNUC ,SDATA=CSA	**Default**: SDATA=DEFS

,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
,ASYNC=YES	
,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).
,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 IEATDUMP macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

DCBAD=*dcbad*
DCB=*dcb*
DSNAD=*dsnad*
DSN=*dsn*
A required input parameter.

DCBAD=*dcbad*
A 4 byte field which contains the address of a previously opened

MACRF=W data control block (DCB) for the data set that is to contain the dump. If specified, the DCB must be writeable in the caller's key.

To code: Specify the RS-type address, or address in register (2)-(12), of a pointer field.

DCB=*dcb*
A parameter of a previously opened MACRF=W data control block (DCB) for the data set that is to contain the dump. If specified, the DCB must be writeable in the caller's key.

To code: Specify the RS-type address, or address in register (2)-(12), of a 96-character field.

DSNAD=*dsnad*
A 4 byte field which contains the area of the name pattern used to create the data set that is to contain the dump. The format of the area is a single byte specifying the length of the name pattern followed by the name pattern itself. The name pattern must resolve to a valid data set name which can 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.

DSN=*dsn*
A parameter of a 101 character input of the area that contains the name pattern used to create the data set that is to contain the dump. The format of the area is a single byte specifying the length of the name pattern followed by the name pattern itself. The name pattern is similar to that used by the operator DUMPDS NAME=parameter, and must not be greater than 100 bytes long. There is no default name pattern available, but the use of system symbols is supported. The name pattern must resolve to a valid data set name which can be allocated from the caller's address space.

To code: Specify the RS-type address, or address in register (2)-(12), of a 45-character field.

,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 parameter of an 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.

To code: Specify the RS-type address, or address in register (2)-(12), of a 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 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: S pecify the RS-type address, or address in register (2)-(12), of a 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 4 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 8 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 8 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 inaccessable 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.

,SDATA=GRSQ
global resource serialization (ENQ/DEQ/RESERVE) queues.

,SDATA=LPA
link pack area for this job.

,SDATA=LSQA
local system queue area.

,SDATA=NUC
non-page-protected areas of the DAT-on nucleus.

,SDATA=RGN
entire private area.

,SDATA=SQA
system queue area.

,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
,ASYNC=YES
An optional parameter that specifies whether the transaction dump should be taken synchronously or asynchronously. The default is ASYNC=NO.

,ASYNC=NO
The transaction dump should be taken synchronously.

,ASYNC=YES
The transaction dump should be taken asynchronously.

,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 45. 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. 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.

Table 45. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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 DCB(AD) of DSN(AD) keyword was specified. **Action**: Reissue IEATDUMP with either the DCB(AD) or DSN(AD) keyword.
00000008	00000009	**Meaning**: Both DCB(AD) and DSN(AD) keywords were specified. **Action**: Reissue IEATDUMP with either the DCB(AD) or DSN(AD) keyword.
00000008	0000000A	**Meaning**: The ALET for the DCB(AD) keyword was not valid. **Action**: Ensure that the access register for the DCB(AD) has a valid ALET when the transaction dump is requested. Reissue IEATDUMP.
00000008	0000000B	**Meaning**: The DCB(AD) was not addressable. **Action**: Ensure that the entire DCB(AD) is addressable using the specified address (and ALET if running in AR ASC mode) when the transaction dump is requested. Reissue IEATDUMP.

Table 45. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 45. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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 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	0000001E	**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	0000001F	**Meaning**: The specified PROBDESC(AD) was not valid. **Action**: Reissue IEATDUMP with a valid problem description area.
00000008	00000020	**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	00000021	**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.

Table 45. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
00000008	00000022	**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	00000023	**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	00000024	**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	00000025	**Meaning**: The specified SUBPLST(AD) was not valid. An invalid subpool was specified. **Action**: Reissue IEATDUMP with a valid subpool list.
00000008	00000026	**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.
00000008	00000027	**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	00000028	**Meaning**: The specified DSPLIST(AD) was not valid. An invalid dataspace was specified. **Action**: Reissue IEATDUMP with a valid dataspace list.
00000008	00000029	**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	0000002A	**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	0000002B	**Meaning**: The specified ECB(AD) was not valid. The ECB was not on a fullword boundary. **Action**: Reissue IEATDUMP with a ECB.
00000008	0000002C	**Meaning**: The dump was rejected because caller authorization insufficient for requested function(s). **Action**: Verify authorization and requested functions. Reissue IEATDUMP.

Table 45. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
00000008	0000002D	**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	0000002E	**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.
00000008	0000002F	**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	00000030	**Meaning**: Unable to create dataspace to capture transaction dump. **Action**: Remedy cause of DSPSERV CREATE failure or request transaction dump specifying DCB(AD). Reissue IEATDUMP.
00000008	00000031	**Meaning**: Unable to add transaction dump data space to access list. **Action**: Remedy cause of ALESERV ADD failure or request transaction dump specifying DCB(AD). Reissue IEATDUMP.
00000008	00000032	**Meaning**: Unable to allocate transaction dump data set. **Action**: Look at allocation failure messages. Reissue IEATDUMP.
00000008	00000033	**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	00000034	**Meaning**: An I/O error occurred writing to the dump. **Action**: Ensure that the specified DCB is MACRF=W and is open when the transaction dump is requested. Reissue IEATDUMP.
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.
0000000C	00000003	**Meaning**: Unable to obtain storage for transaction dump from subpool 239 above the line. **Action**: Determine why storage is not available and reissue IEATDUMP.

Table 45. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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	000000FF	**Meaning**: IEAVTDMP's recovery received control. **Action**: Inform the system programmer.
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 DCB:

```
         OPEN (DUMPDCB,OUTPUT)
         IEATDUMP DCB=DUMPDCB,HDR=DUMPTITL
         CLOSE DUMPDCB
         .
         .
         .
DUMPDCB  DCB DDNAME=TDUMP,MACRF=(W),BLKSIZE=4160,LRECL=4160,   X
               RECFM=FB,DSORG=PS
DUMPTITL DC    AL1(E1-S1)
S1       DC    C'TRANSACTION DUMP TO AN OPEN DCB'
E1       EQU   *
```

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   *
```

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:

Minimum authorization: If auth_level=AUTHORIZED, supervisor state and PSW key 0. If auth_level=UNAUTHORIZED, any state and 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
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

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.

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 72-byte register save area.

© Copyright IBM Corp. 1988, 2002

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

```
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

Indicates the caller's authorization level of the pause element being allocated. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	When using the allocated pause element via other services, any auth_level value will be sufficient.
IEA_AUTHORIZED	1	When using the allocated pause element via other services, auth_level=IEA_AUTHORIZED will be required. Caller must be both key 0 and supervisor state.

,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.

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. 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 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.
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.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning:** This service routine encountered an unexpected error. The system rejects this service request. **Action:** Contact IBM support.

IEAVDPE — Deallocate_Pause_Element

Description

Deallocate_Pause_Element frees a pause element that is no longer needed.

Environment

The requirements for the caller are:

Minimum authorization: If auth_level=AUTHORIZED, supervisor state and PSW key 0. If auth_level=UNAUTHORIZED, any state and PSW key.
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 31-bit addressing mode.
ASC mode: Primary mode.
Interrupt status: Enabled
Locks: No locks held.
Control parameters: Must 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

When the calling program specifies aut_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

© Copyright IBM Corp. 1988, 2002

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.

```
                    |
                    |(return_code
CALL  IEAVDPE       |,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 (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The pause element being deallocated must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	The pause element being deallocated may have been allocated with any auth_level. Caller must be both key 0 and supervisor state.

,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)	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.
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. **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
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 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:** Contact IBM support.

IEAVPSE — Pause Service

Description

Call Pause to make the current task or SRB nondispatchable. Once 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:

Minimum authorization: If auth_level=AUTHORIZED, supervisor state and PSW key 0. If auth_level=UNAUTHORIZED, any state and PSW key.
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 31-bit addressing mode.
ASC mode: Primary mode.
Interrupt status: Enabled
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

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.

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).

© Copyright IBM Corp. 1988, 2002

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

```
                    |
                    |(return_code
                    |,auth_level
CALL  IEAVPSE       |,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 authorization level of the pause element being paused. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (hexadecimal)	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 may have been allocated with any auth_level. Caller must be 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 being used to pause the current task or SRB. 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 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.

,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.

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.
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 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:** Contact IBM support.

IEAVRLS — 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:

Minimum authorization: If auth_level=AUTHORIZED, supervisor state and PSW key 0. If auth_level=UNAUTHORIZED, any state and PSW key.

Dispatchable unit mode: Task or SRB

Cross memory mode: Any PASN, any HASN, any SASN

AMODE: 31-bit addressing mode.

ASC mode: Primary mode.

Interrupt status:
- Unauthorized caller - enabled for I/O and external interrrupts.
- Authorized caller - enabled or disabled for I/O and external interrupts.

Locks:
- No locks held.
- Authorized caller can hold the CPU or local 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 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

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release 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.

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).

© Copyright IBM Corp. 1988, 2002

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

```
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 of the pause element being released. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which can be used by the calling program. The following levels are supported.

Variable	Value (hexadecimal)	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 may have been allocated with any auth_level. Caller is both key 0 and supervisor state.

,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 (that is, the task or SRB has not been paused), the task or SRB will not be paused when it attempts to do so. 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 or SRB. 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.

Return code in: Decimal (Hex)	Equate symbol	Meaning and Action
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; 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.

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 specified auth_level=UNAUTHORIZED, but a pause element token allocated with auth_level=AUTHORIZED was specified. 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 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.

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 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:

Minimum authorization: None.
Dispatchable unit mode: Task or SRB
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 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:

© Copyright IBM Corp. 1988, 2002

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

```
                          |
                          |(return_code
                          |,auth_level
                          |,pause_element_token
CALL  IEAVRPI             |,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 of the creator of the pause element specified by the input PET. One of the following values:

Constant Name	Value (hexadecimal)	Meaning
IEAV_PET_UNAUTHORIZED	0	The pause element was allocated with auth_level=IEA_UNAUTHORIZED.
IEAV_PET_AUTHORIZED	1	The pause element was allocated with auth_level=IEA_AUTHORIZED.

,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.

State Constant Hexadecimal (Decimal)	Meaning
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.

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.
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 auth_level=IEA_UNAUTHORIZED, but specified a pause element token allocated with 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 caller specified auth_level=IEA_UNAUTHORIZED, but specified a token of a 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**: Contact IBM support.

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:

Minimum authorization: None.
Dispatchable unit mode: Task or SRB
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

© Copyright IBM Corp. 1988, 2002

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

```
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.

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.

IEAVXFR — 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:

Minimum authorization: If auth_level=AUTHORIZED, supervisor state and PSW key 0. If auth_level=UNAUTHORIZED, any state and PSW key.
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 31-bit addressing mode.
ASC mode: Primary mode.
Interrupt status: Enabled
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

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 IEA_UNAUTHORIZED,

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:

© Copyright IBM Corp. 1988, 2002

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

```
                    |
                    |(return_code
                    |,auth_level
                    |,current_du_pause_element_token
CALL  IEAVXFR       |,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 specified pause element. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which can be used by the calling program. The following levels are supported:

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The specified pause elements must have been allocated with auth_level=IEA_UNAUTHORIZED.
IEA_AUTHORIZED	1	The specified pause elements may have been allocated with any auth_level. Caller must be both key 0 and supervisor state.

,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. 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 update_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 or SRB 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 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_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.

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.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
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. 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.
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 specified auth_level=UNAUTHORIZED, but a pause element token allocated with auth_level=AUTHORIZED was specified. 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 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.
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**: Contact IBM support.

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:

Minimum authorization:	Supervisor state or 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

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IEECMDS.
IEECMDS	
b	One or more blanks must follow IEECMDS.

REQUEST=**COUNT** ,REQUEST=INFO ,REQUEST=REMOVE	**Default:** REQUEST=COUNT
,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 *bufsize*: RS-type address or address in register (2) - (12).
,CLASS=*class* ,CLASS=**ANY_CLASS**	*class*: RS-type address or address in register (2) - (12). **Default:** CLASS=ANY_CLASS
,CMD=*cmd* ,CMD=**ANY_CMD**	*cmd*: RS-type address or address in register (2) - (12) **Default:**CMD=ANY_CMD
,ID =*id* ,ID=**ANY_ID**	*id*: RS-type address or address in register (2) - (12) **Default:**: ID=ANY_ID
,JOB=*job* ,JOB=**ANY_JOB**	*job*: RS-type address or address in register (2) - (12) **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** ,PLISTVER=MAX ,PLISTVER=1	**Default**: PLISTVER=IMPLIED_VERSION

,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 on command classes, see the description of comand 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.

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
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.
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.

Hexadecimal Return Code	Hexadecimal Reason Code	Equate Symbol Meaning and Action
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 and/or BUFSIZE 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.
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.

IEEQEMCS — Query EMCS Console

Description

This macro returns information about EMCS consoles in the system.

Environment

The requirements for the caller are:

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

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name:* symbol. Begin *name* in column 1.
b	One or more blanks must precede IEEQEMCS.
IEEQEMCS	
b	One or more blanks must follow IEEQEMCS.

REQUEST=**COUNT** REQUEST=SUMMARY REQUEST=INFO REQUEST=FULL	**Default**: REQUEST=COUNT
,STATUS=**ACTIVE** ,STATUS=INACTIVE ,STATUS=ALL ,STATUS=BACKLOG ,STATUS=ERR	**Default**: STATUS=ACTIVE
,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** ,AUTH=MASTER ,AUTH=SYS ,AUTH=IO ,AUTH=CONS ,AUTH=ALL ,AUTH=INFO ,AUTH=SYSONLY ,AUTH=IOONLY	**Default**: AUTH=ANY

,AUTH=CONSONLY	
,AUTH=ALLONLY	
,AUTH=INFOONLY	
,ATTR=**ANY**	**Default**: ATTR=ANY
,ATTR=YES	
,ATTR=ROUT	
,ATTR=UD	
,ATTR=UDN	
,ATTR=HC	
,ATTR=AUTO	
,ATTR=AUTON	
,ATTR=MN	
,ATTR=NONE	
,DOM=**ANY**	**Default**: DOM=ANY
,DOM=YES	
,DOM=NORMAL	
,DOM=ALL	
,DOM=NONE	
,BUFPTR=*bufptr*	*bufptr:* RS-type address or address in register (2) - (12).
,BUFSIZE=*bufsize*	*bufsize:* RS-type address or address in register (2) - (12).
,CONSCNT=*conscnt*	*conscnt:* 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).
,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*)	
,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.

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 charaters.

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=UD
,ATTR=UDN
,ATTR=HC
,ATTR=AUTO
,ATTR=AUTON
,ATTR=MN
,ATTR=NONE
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=UD
Return consoles receiving UD messages.

,ATTR=UDN
Return consoles not receiving UD messages.

,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.

,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.

,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.

,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.

,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.

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.

,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 46. 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.

Table 46. Return and Reason Codes for the IEEQEMCS Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
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.
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.

Table 46. Return and Reason Codes for the IEEQEMCS Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
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.
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.

Table 46. Return and Reason Codes for the IEEQEMCS Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
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 dataspace 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
          LA   REG9,ECDM_TYPE_CNSW    Load the type of block in R9
          CH   REG9,0(REG8)           Is this a CNSWITCH block?
          BE   CNSWBLK                Yes, branch to CNSWITCH 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
CNSWBLK   LR   CNSWREG,REG8           Load CNSW block addr into R7
          A    REG8,ECDM_CNSW_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
```

IEEVARYD — Vary Device(s) Online and 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 and z/OS.e 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:

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.

Restrictions

None.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEEVARYD.
IEEVARYD	
b	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).

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 message(s) 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

None.

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 should take.

Table 47. 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.
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.

Table 47. Return and Reason Codes for the IEEVARYD Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEEVARYD.
IEEVARYD	
b	One or more blanks must follow IEEVARYD.

MF=(L,*list addr*) MF=(L,*list addr,attr*) MF=(L,*list addr*,0D)	*list addr*: symbol. *attr*: 1- to 60-character input string. **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:

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*) ,MF=(E,*list addr*,COMPLETE)	*list addr*: RX-type address or register (2) - (12). **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.

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 on 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:

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:

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFPPSCN.
IEFPPSCN	
b	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*	*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).

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, Vol 3 (IVT-RCWK)* 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 48. 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:

---

name — *name*: Symbol. Begin *name* in column 1.

b	One or more blanks must precede IEFPPSCN.
IEFPPSCN	
b	One or more blanks must follow IEFPPSCN.

,MF=(L,*cntl)* ,MF=(L,*cntl,attr)* ,MF=(L,*cntl*,0D)	*cntl*: Symbol. *attr*: 1- to 60-character input string. **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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFPPSCN.
IEFPPSCN	
b	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*	*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)* ,MF=(E,*cntl*,COMPLETE)	*cntl*: RX-type address or register (2) - (12). **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.

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:

Minimum authorization:	Supervisor state, and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=SASN=HASN
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.

© Copyright IBM Corp. 1988, 2002

	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:

---

name	*name*:
b	One or more blanks must precede IEFQMREQ.
IEFQMREQ	
b	One or more blanks must follow IEFQMREQ.

Parameters

For information on 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 49. 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 50. Reason Codes for the IEFQMREQ Macro

Hexadecimal Reason Code	Meaning
04	**Meaning**: The function you requested was not valid.

Table 50. Reason Codes for the IEFQMREQ Macro (continued)

Hexadecimal Reason Code	Meaning
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.

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 467 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 471 for the syntax of this request.
- IEFSSI REQUEST=OPTIONS, which defines a set of optional subsystem characteristics. See 474 for the syntax of this request.
- IEFSSI REQUEST=DEACTIVATE, which deactivates a subsystem. See 477 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 480 for the syntax of this request.
- IEFSSI REQUEST=PUT, which stores subsystem-defined data for the subsystem. See 483 for the syntax of this request.
- IEFSSI REQUEST=GET, which retrieves subsystem-defined data previously stored using the IEFSSI REQUEST=PUT service. See 486 for the syntax of this request.
- IEFSSI REQUEST=QUERY, which obtains information about a currently defined subsystem. See 489 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

© Copyright IBM Corp. 1988, 2002

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 on using dynamic subsystem services, see *z/OS MVS Using the Subsystem Interface*. This book also includes information about related macros IEFSSVT and IEFSSVTI.

Environment

The requirements for the caller are:

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 (IEAVCVT) 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSI.
IEFSSI	
b	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=*NO_INITPARM*	*initparm:* RS-type address or address in register (2) - (12). **Default**: INITPARM=NO_INITPARM
,INITPLEN=*initplen*	*initplen*: RS-type address or address in register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default**: IMPLIED_VERSION *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=*NULL*	*com*: comment string **Default**: COM=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)	**Default:** MF=S

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 master console.

,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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSI.
IEFSSI	
b	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
,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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSI.
IEFSSI	
b	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* ,COMMAND=*YES*	**Default**: COMMAND=NO
,REQDSUB=*MSTR* ,REQDSUB=*PRI*	**Default**: REQDSUB=MSTR
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default**: IMPLIED_VERSION *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=*NULL*	*com*: comment string **Default**: COM=NULL
,MF=S ,MF=(L,*list addr*) ,MF=(L,*list addr,attr*) ,MF=(L,*list addr*,0D) ,MF=(E,*list addr*)	**Default:** MF=S

,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 should 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 for REQDSUB=MSTR and REQDSUB=PRI 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.

,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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSI.
IEFSSI	
b	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 ,PLISTVER=*plistver*	 **Default**: IMPLIED_VERSION *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=*NULL*	*com*: comment string **Default**: COM=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)	**Default:** MF=S

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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSI.
IEFSSI	
b	One or more blanks must follow IEFSSI.

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 ,PLISTVER=*plistver*	**Default**: IMPLIED_VERSION *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=*NULL*	*com*: comment string **Default**: COM=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)	**Default:** MF=S

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.

Once 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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSI.
IEFSSI	
b	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 ,PLISTVER=*plistver*	**Default**: IMPLIED_VERSION *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=*NULL*	*com*: comment string **Default**: COM=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)	**Default:** MF=S

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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IEFSSI.
IEFSSI	
ƀ	One or more blanks must follow IEFSSI.

SUBNAME=*subname*	*subname*: RS-type address or address in register (2) - (12).
,REQUEST=GET	

,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 ,PLISTVER=*plistver*	**Default**: IMPLIED_VERSION *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=*NULL*	*com*: comment string **Default**: COM=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)	**Default:** MF=S

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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSI.
IEFSSI	
b	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.
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default**: IMPLIED_VERSION *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=*NULL*	*com*: comment string **Default**: COM=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)	**Default:** MF=S

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 quotes 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 51. 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 51. 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 useable 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.

Table 51. Return and Reason Codes for the IEFSSI Macro (continued)

Return Code decimal (hex)	Reason Code decimal (hex)	Meaning and Action
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.
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)	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.

Table 51. Return and Reason Codes for the IEFSSI Macro (continued)

Return Code decimal (hex)	Reason Code decimal (hex)	Meaning and Action
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.

```
         IEFSSI REQUEST=OPTIONS,SUBNAME=SNAME,COMMAND=YES,REQDSUB=PRI, X
              RETCODE=RETURN_CODE,RSNCODE=REASON_CODE
```

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
```

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 page 501 for the syntax of this request.
- IEFSSVT REQUEST=DISABLE, which disables supported function codes. See page 505 for the syntax of this request.
- IEFSSVT REQUEST=ENABLE, which enables additional function codes. See page 507 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 page 511 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.

© Copyright IBM Corp. 1988, 2002

For information on using dynamic subsystem services, see *z/OS MVS Using the Subsystem Interface*. This book also includes information about related macros IEFSSVTI and IEFSSI.

Environment

The requirements for the caller are:

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 (IEAVCVT) 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 on dynamic SSI services.

Syntax for REQUEST=CREATE

The syntax of the IEFSSVT REQUEST=CREATE macro is written as follows:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVT.
IEFSSVT	
b	One or more blanks must follow IEFSSVT.

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
,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 on 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.

,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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVT.
IEFSSVT	
b	One or more blanks must follow IEFSSVT.

SUBNAME=subname	
,REQUEST=DISABLE	
,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 (OUTTTOKEN 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 quotes 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:

---

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=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 quotes 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVT.
IEFSSVT	
b	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 ,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	
,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 quotes 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 Symbols
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 52. Return and Reason Codes for the IEFSSVT Macro

Return Code Decimal (hex)	Reason Code Decimal (hex)	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.
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.

Table 52. Return and Reason Codes for the IEFSSVT Macro (continued)

Return Code Decimal (hex)	Reason Code Decimal (hex)	Meaning and Action
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)	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.

Table 52. Return and Reason Codes for the IEFSSVT Macro (continued)

Return Code Decimal (hex)	Reason Code Decimal (hex)	Meaning and Action
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.
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
```

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.

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 page 521 for the syntax of this request.
- IEFSSVTI TYPE=INITIAL, which begins the definition of a static function routine input table. See page 521 for the syntax of this request.
- IEFSSVTI TYPE=ENTRY, which defines a function routine entry in a static input table. See page 522 for the syntax of this request.
- IEFSSVTI TYPE=FINAL, which ends the definition of a static function routine input table. See page 524 for the syntax of this request.
- IEFSSVTI TYPE=SET, which modifies a function routine entry in an existing input table. See page 525 for the syntax of this request.
- IEFSSVTI TYPE=RESERVE, which reserves storage for a function routine input table. See page 527 for the syntax of this request.
- IEFSSVTI TYPE=COPY, which copies a function routine input table. See page 528 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

© Copyright IBM Corp. 1988, 2002

- 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 on using dynamic subsystem services, see *z/OS MVS Using the Subsystem Interface*. This book also includes information about the related macro IEFSSI.

Environment

The requirements for the caller are:

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.

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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
b	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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
b	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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
b	One or more blanks must follow IEFSSVTI.

TYPE=ENTRY	
,FUNCNAME=*funcname* ,FUNCADDR=*funcaddr*	*funcname*: RS-type address or address in register (2) - (12). *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=*0*	*fcodes*: RS-type address or address in register (2) - (12). **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 on 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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
b	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:

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* ,FUNCADDR=*funcaddr*	*funcname*: RS-type address or address in register (2) - (12). *funcaddr*: RS-type address or address in register (2) - (12). **Default**: none
,FUNCAMODE=*HOB* ,FUNCAMODE=*31* ,FUNCAMODE=*24*	**Default**: FUNCAMODE=HOB **Note:** FUNCAMODE is valid with FUNCADDR.
,FCODES=*fcodes(,fcodes...)* ,FCODES=*0*	*fcodes*: RS-type address or address in register (2) - (12). **Default**: FCODES=0

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 on 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:

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=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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IEFSSVTI.
IEFSSVTI	
b	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.

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:

Notes:

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
```

IOCINFO — Obtain MVS I/O Configuration Information

Description

Use the IOCINFO macro to obtain the following I/O configuration information:
- I/O configuration token
- The maximum device measurement block index that is currently assigned

Environment

The requirements for the caller are:

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:

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOCINFO.
IOCINFO	
b	One or more blanks must follow IOCINFO.

IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,MAXMBI=*maxmbi addr*	*maxmbi addr*: RS-type address or register (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default**: SYSTEM
,RETCODE=*retcode addr*	*retcode addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).

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.

,MAXMBI=*maxmbi addr*
Specifies the address of a halfword field where the system returns the maximum device measurement block index that is currently assigned.

,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.

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.
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.
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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOCINFO.
IOCINFO	
b	One or more blanks must follow IOCINFO.

MF=(L,*list addr*) MF=(L,*list addr*,*attr*) MF=(L,*list addr*,0D)	*list addr*: Symbol. *attr*: 1- to 60- character input string **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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOCINFO.
IOCINFO	
b	One or more blanks must follow IOCINFO.

IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,MAXMBI=*maxmbi addr*	*maxmbi addr*: RS-type address or register (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default**: SYSTEM
,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*) ,MF=(E,*list addr*,COMPLETE)	*list addr*: RX-type address or register (2) - (12). **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.

IOSADMF — Transfer HiperspaceData

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:

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.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSADMF.
IOSADMF	
b	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.
,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 53. 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 7, you can code the RANGLIST parameter and NUMRANGE parameter as follows:

```
NUMRANGE=3 ,RANGLIST=(5)

          or

NUMRANGE=3, RANGLIST=RANGADDR
```

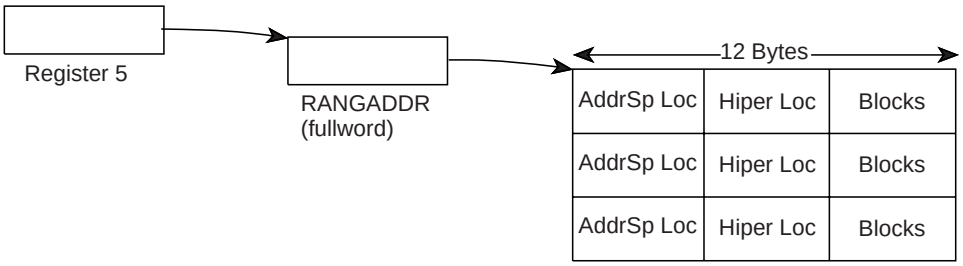


Figure 7. RANGLIST and NUMRANGE Parameters

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 move-page 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 54. 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 due to a communication error. The request was started, but the system stopped the request due to 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 due to a communication error. The request was started, but the system stopped the request due to 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 54. 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.

Table 54. Return and Reason Codes for the IOSADMF Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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.
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 page “Environment” on page 539.
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.

Table 54. Return and Reason Codes for the IOSADMF Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 page “Environment” on page 539.
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.
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 re-issue 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.

Table 54. Return and Reason Codes for the IOSADMF Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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:

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*) ,MF=(L,*list addr,attr*) ,MF=(L,*list addr*,0D)	*list addr*: symbol. *attr*: 1- to 60-character input string. **Default**: 0D

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSADMF.
IOSADMF	
b	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* ,NUMRANGE=*num-addr*	*n*: Decimal digit from 1 to 125. *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*) ,MF=(E,*list addr*,COMPLETE)	*list addr*: RX-type address or address in register (2) - (12). **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.

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:

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 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.

© Copyright IBM Corp. 1988, 2002

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:

---

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.

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 a UCB Function” on page 555, “Release a Captured UCB Function” on page 560, and “Translate Captured to Actual Address Function” on page 564.

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:

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)

© Copyright IBM Corp. 1988, 2002

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 a UCB Function

Syntax

The standard form of the IOSCAPU macro with the CAPTUCB option is written as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	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 ,LASTING=YES	**Default**: LASTING=NO
,CAPTCOM=NO ,CAPTCOM=YES ,CAPTCOM=NEVER	**Default**: CAPTCOM=NO
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default**: LINKAGE=SYSTEM
,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 55. 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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	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 ,LASTING=YES	**Default**: LASTING=NO
,CAPTCOM=NO ,CAPTCOM=YES	**Default**: CAPTCOM=NO
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default**: LINKAGE=SYSTEM
,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*) ,MF=(E,*list addr*,COMPLETE) ,MF=(E,*list addr*,NOCHECK)	*list addr*: RX-type address or address in register (2) - (12). **Default**: COMPLETE

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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	One or more blanks must follow IOSCAPU.

UCAPTUCB	
,CAPTPTR=*captptr*	*captptr* :RS-type or address in register (2) - (12).
,LASTING=NO ,LASTING=YES	**Default**: LASTING=NO
,CAPTCOM=NO ,CAPTCOM=YES	**Default**: CAPTCOM=NO
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default**: LINKAGE=SYSTEM
,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 56. 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.

Table 56. Return and Reason Codes for the IOSCAPU UCAPTUCB Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 UCB that resides above 16 megabytes. **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 writtten as follows:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	One or more blanks must follow IOSCAPU.

MF=(L,*list addr*) MF=(L,*list addr,attr*) MF=(L,*list addr*,0D)	*list addr*: symbol. *attr*: 1- to 60-character input string. **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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	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 ,LINKAGE=BRANCH	**Default**: LINKAGE=SYSTEM
,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 57. 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:

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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCAPU.
IOSCAPU	
b	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 ,ASID=*asid*	**Default**: ASID=CURRENT *asid* :RS-type or address in register (2) - (12).
,LINKAGE=SYSTEM ,LINKAGE=BRANCH	**Default**: LINKAGE=SYSTEM
,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*) ,MF=(E,*list addr*,COMPLETE) ,MF=(E,*list addr*,NOCHECK)	*list addr*: RX-type address or address in register (2) - (12). **Default**: COMPLETE

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.

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 information that IOSCDR retrieves is returned in an area called a configuration data record (CDR), which is mapped by the mapping macro IHACDR.

The format of IHACDR is in *z/OS MVS Data Areas, Vol 1 (ABEP-DALT)*. For more information about the contents of CDRs, see *ESA/390 Common I/O Device Commands*.

Environment

The requirements for the caller are:

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 CDR. When IOSCDR completes, it unpins the device.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSCDR.
IOSCDR	
ƀ	One or more blanks must follow IOSCDR.

DEVN=*device num*	*device num:* RX-type address or address in 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 ,LINKAGE=LINK	**Default**: LINK
,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).
,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).

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.

,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, Vol 2 (DCCB-ITZYRETC)* 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.

,READ=NOIO
,READ=IO
,READ=COND

Specifies that IOSCDR retrieves the CDR in one of the following ways:

- NOIO—IOSCDR retrieves the last CDR known to MVS for the specified device and path. Note that this CDR 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 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 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 in the shortest time possible.

,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 zeroes, 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*.

,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.

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.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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.
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.
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.

First, ensure that the program is running on an MVS/ESA SP 5.1 system. 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. Once 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 on 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
*
*.................................................................*
*                                                                 *
*       Ensure that this program is running on an MVS/ESA SP 5.1  *
*       system by checking the version information in the CVT.    *
*                                                                 *
*.................................................................*
              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
```

IOSCHPD — IOS CHPID Description Service

Description

The IOSCHPD macro returns the acronym and/or description of a channel path (CHP) type.

Environment

The requirements for the caller are:

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
- the 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

© Copyright IBM Corp. 1988, 2002

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:

---

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

CHPID=*chpid*	*chpid*: RS-type address or register (2) - (12).
,ATTR=*attr*	*attr*: RS-type address or register (2) - (12).
CHP_TYPE=*chp_type*	*chp_type*: RS-type address register (2) - (12).
,CHP_PARM=*chp_parm*	*chp_type*: RS-type address or register (2) - (12).
,CHP_PARM=**0**	**Default**: 0
,ACRONYM=*acronym*	*acronym*: RS-type address or register (2) - (12).
,DESC=*desc*	*desc*: RS-type address or register (2) - (12).
,LINKAGE=**SYSTEM** ,LINKAGE=BRANCH	**Default**: LINKAGE=SYSTEM
,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=1	**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**)	**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 IOSCHPD macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

CHPID=*chpid*
CHP_TYPE=*chp_type*
A required input parameter.

CHPID=*chpid*
A parameter which specifies the CHPID number for which to retrieve the acronym and/or description.

If the CHPID is defined as a managed channel path, then 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.

CHP_TYPE=*chp_type*
A parameter which specifies the channel path type for which to retrieve the acronym and/or description. 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).

To code: Specify the RS-type address, or address in register (2)-(12), of a one-byte field.

ATTR=*attr*
An optional input parameter, used with CHPID=*chpid* parameter, that specifies the 32-character 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.

CHP_PARM=*chp_parm*
CHP_PARM=0
An optional input parameter, used 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, then 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.

,ACRONYM=*acronym*
,DESC=*desc*
A required output parameter. One or more of these parameters may be specified.

,ACRONYM=*acronym*
A parameter area which 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*
A parameter area which is to receive the description.

To code: Specify the RS-type address, or address in register (2)-(12), of a 32-character field.

,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 which 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
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 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 58. Return and Reason Codes for the IOSCHPD Macro

Hexadecimal Return Code	Reason Codes, Meaning and Action
00	The acronym and/or description has been returned.

Table 58. Return and Reason Codes for the IOSCHPD Macro (continued)

Hexadecimal Return Code	Reason Codes, Meaning and Action
04	The acronym and/or description have not been returned (the acronym and description output areas have been set to zeroes). **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.
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 — The ATTR= keyword can only be specified with CHPID=.
0C	Recovery was entered.
20	Recovery was entered.

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:

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.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCMXA.
IOSCMXA	
b	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).
,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. See “Alternative List Form Macros” on page 12 for further information.

Syntax

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

---

name — *name*: symbol. Begin *name* in column 1.

b — One or more blanks must precede IOSCMXA.

IOSCMXA

b	One or more blanks must follow IOSCMXA.

MF=(L,*list addr*) MF=(L,*list addr,attr*) MF=(L,*list addr*,0D)	*list addr*: symbol. *attr*: 1- to 60-character input string. **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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCMXA.

IOSCMXA

b	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*) ,MF=(E,*list addr*,COMPLETE)	*list addr*: RX-type address or address in register (2) - (12). **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.

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:

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).

© Copyright IBM Corp. 1988, 2002

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:

---

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.

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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSCUMOD.
IOSCUMOD	
b	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.
&MASK2=*mask2*	*mask2*: 2-byte hex symbol.
&MASK3=*mask3*	*mask3*: 2-byte hex symbol.
&MASK4=*mask4*	*mask4*: 2-byte hex symbol.
&DCM_SUPPORTED=**YES** &DCM_SUPPORTED=NO	**Default**: YES

© Copyright IBM Corp. 1988, 2002

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.

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 (DCE) segment. For example, the DASD DCE is mapped by the IECDDCE macro and the tape DCE is mapped by the IECUCBCX macro.

Note: If you supply a captured UCB as input, you receive the address of the captured UCB DCE except under either of the following conditions:

- The unit information module (UIM) indicates that the DCE can reside above 16 megabytes independent of the rest of the UCB.
- The UIM indicates that a single DCE can be shared by multiple UCBs.

In these cases, you receive the address of the actual, not captured, UCB DCE.

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:

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

© Copyright IBM Corp. 1988, 2002

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 DCE if the UCB has a DCE. Zero if the UCB does not have a DCE.
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:

---

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.

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.

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:

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)

© Copyright IBM Corp. 1988, 2002

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:

---

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.
,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 59. 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 60. 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.

Table 60. Return and Reason Codes for the IOSENQ Macro (continued)

Hexadecimal Return Code	Reason Codes, Meaning and Action
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.

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 (IEAVCVT) 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:

---

name — *name*: Symbol. Begin *name* in column 1.

b — One or more blanks must precede IOSINFO.

IOSINFO

© Copyright IBM Corp. 1988, 2002

b	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
```

IOSLOOK — Locate Unit Control Block

Note: The UCBLOOK macro is the preferred programming interface.

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:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSLOOK.
IOSLOOK	
b	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.

© Copyright IBM Corp. 1988, 2002

Return Codes

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

Table 61. 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)
```

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:

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), 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.

© Copyright IBM Corp. 1988, 2002

Output Register Information

Before issuing the IOSPTHV macro, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSPTHV.
IOSPTHV	
b	One or more blanks must follow IOSPTHV.

,DEVN=*device number*	*device number*: RX-type address or address in register (2) - (12).
,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 zeroes, 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 ths 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.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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. First, ensure that the program is running on an MVS/ESA SP 5.1 system. 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. Once 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
*
*.....................................................................*
*                                                                     *
*        Ensure that this program is running on an MVS/ESA SP 5.1     *
*        system by checking the version information in the CVT.       *
*                                                                     *
*.....................................................................*
              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 IOSCDR  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
```

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:

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

© Copyright IBM Corp. 1988, 2002

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:

---

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. See “Alternative List Form Macros” on page 12 for further information.

Syntax

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

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*) MF=(L,*list addr,attr*) MF=(L,*list addr*,0D)	*list addr*: symbol. *attr*: 1- to 60-character input string. **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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IOSUPFA.
IOSUPFA	
b	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*) ,MF=(E,*list addr*,COMPLETE)	*list addr*: RX-type address or address in register (2) - (12). **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.

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:

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.

© Copyright IBM Corp. 1988, 2002

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:

---

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.

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:

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)

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IOSWITCH.
IOSWITCH	
ƀ	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	
,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=1	**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**)	**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 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 called. 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*
A required halfwood input paramater that identifies the subpool to be used for obtaining storage for the switch data area. Valid subpools are in common storage: 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) reason code.

The following table identifies the hexadecimal return and reason codes:

Table 62. Return and Reason Codes for the IOSWITCH Macro

Hexadecimal Return Code	Reason Codes, Meaning and Action
00	IOSWITCH completed successfully.

Table 62. Return and Reason Codes for the IOSWITCH Macro (continued)

Hexadecimal Return Code	Reason Codes, Meaning and Action	
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.
08	Program error.	
	Reason Code	**Meaning/Action**
	01	**Meaning:** The subpool provided by the caller is not in common storage. **Action:** Correct the subpool and reissue the macro.
	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 macro.
	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.
20	System error. An unexpected error occurred.	

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:

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 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.

© Copyright IBM Corp. 1988, 2002

Performance Implications

None.

Syntax

The IRDFSD macro is written as follows:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IRDFSD.
IRDFSD	
b	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*	
FSADDRESS=*fsaddress*	
,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 device(s) 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 interogated.

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*
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.

,FSADDRESS=*fsaddress*
A required input parameter which contains the address of the storage mapped in IHAF

To code: Specify the RS-type address of address in register (2)-(12), of a 4 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,*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 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 xmfctrl and provides full syntyax checking with the default setting.

,*xmfctrl*
Thisis 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 63. Return Codes for IRDFSD macro

Hexadecimal Return Code	Meaning
00	**Meaning**: Successful completion. Data returned for all FICON switches.
04	**Meaning**: No FICON switch devies found. No FSD area was obtained.
08	**Meaning**: I/O errors occured. SOme switches did not return data.
0C	**Meaning**: I/O errors occured. No data obtained for any devices.
10	**Meaning**: Unexpected error.

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:

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.

© Copyright IBM Corp. 1988, 2002

Performance Implications

None.

Syntax

The IRDFSDU macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede IRDFSDU.
IRDFSDU	
b	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** ,PLISTVER=MAX ,PLISTVER=1	**Default**: PLISTVER=IMPLIED_VERSION
,MF=**S** ,MF=(L,*mfctrl*) ,MF=(L,*mfctrl,mfattr*) ,MF=(L,*mfctrl*,**0D**) ,MF=(E,*mfctrl*) ,MF=(E,**COMPLETE**)	**Default**: MF=S

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 64. 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
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.
20	1	**Meaning**: An ESTAE could not be established.

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.

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:

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.

© Copyright IBM Corp. 1988, 2002

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

```
                    |
                    |,(number_of_latches
                    |,latch_set_name
CALL  ISGLCRT       |,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 a value of ISGLCRT_PRIVATE (or a value of 0).

,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 65. 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.

Examples of Calls to Latch Manager Services

The following is an example of how to call all the latch manager services in assembler language.

```
LATCHXMP CSECT
LATCHXMP AMODE 31
LATCHXMP RMODE ANY
         BAKR  R14,R0              save regs
         SAC   0                   ensure primary mode
         LR    R12,R15             establish addressability
         USING LATCHXMP,R12
         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'   indicate regs saved on stack
*
*********************************************************************
* create latch set
*********************************************************************
*
         CALL ISGLCRT,(NUM_LATCH,LS_NAME,PRIVATE,LS_TOKEN,RETCD),      X
               MF=(E,CREATE_DPL)
*
*********************************************************************
* obtain latch
*********************************************************************
*
         MVC  REQ_IDH,PSAAOLD-PSA  use ASCB address and
         MVC  REQ_IDL,PSATOLD-PSA  TCB address as requestor ID
*
         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
*********************************************************************
*
         CALL ISGLREL,(LS_TOKEN,LATCH_TOKEN1,UNCOND,WORK_AREA,RETCD),  X
               MF=(E,RELEASE_DPL)
*
*********************************************************************
* purge requestor from latch set
*********************************************************************
*
         CALL ISGLPRG,(LS_TOKEN,REQ_ID,RETCD),MF=(E,PURGE_DPL)
*
*********************************************************************
* exit
*********************************************************************
*
         LR    R1,R13              load reg1 with dynamic area
         STORAGE RELEASE,ADDR=(R1),LENGTH=DYNALEN free storage
         SLR   R15,R15             set return code to zero
         PR                        restore caller's regs and return;
                                   also restores caller's PSW key and
                                   state without a MODESET MODE=PROB
*
*********************************************************************
* 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
************************************************************************
*
              ISGLMASM              include assembler IDF
*
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
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               register save area
*
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
*
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
*
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, release, and
*                                    purge
LATCH_TOKEN1 DS  2F                latch token - output from obtain
*                                    and input to release
REQ_ID       DS  0D                double word requestor ID - input to
*                                    both obtain and purge
REQ_IDH      DS  F                 first half of requestor ID (ascb@)
REQ_IDL      DS  F                 second half set to zeros
RETCD        DS  F                 return code - output from services
WORK_AREA    DS  32D               256-byte work area
DYNALEN      EQU *-DYNASTORE
*
             IHAPSA                needed for ascb@ and tcb@ req ID
             END LATCHXMP
```

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:

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

1. If you specify an obtain_option of ISGLOBT_ASYNC_ECB (as described in "Parameters" on page 659), 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 should 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
*
```

© Copyright IBM Corp. 1988, 2002

```
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

```
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 “ISGLPRG — Purge a Requestor from a Latch Set” on page 669.

,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 66. 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 “Examples of Calls to Latch Manager Services” on page 654 for an example of how to call Latch_Obtain in assembler language.

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:

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
*
```

© Copyright IBM Corp. 1988, 2002

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

```
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) 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 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 67. 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).

Table 67. ISGLREL Return Codes (continued)

Return code and Equate Symbol	Meaning and Action
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.
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 "Examples of Calls to Latch Manager Services" on page 654 for an example of how to call Latch_Release in assembler language.

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:

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 should 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:

© Copyright IBM Corp. 1988, 2002

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.

```
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 paretheses), the equate symbol associated with each return code, the meaning of each return code, and a recommended action:

Table 68. 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 “Examples of Calls to Latch Manager Services” on page 654 for an example of how to call Latch_Purge in assembler language.

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:

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 should 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
*
```

© Copyright IBM Corp. 1988, 2002

```
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.

```
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 69. 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.

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 on 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:

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:

---

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede ITTFMTB.
ITTFMTB	

© Copyright IBM Corp. 1988, 2002

b	One or more blanks must follow ITTFMTB.

MAP TABLEDATA=*tabname* EVENTDATA=*eventid* TABLEEND	Required choice. Select one of four options. *tabname*: Symbol up to eight characters long. *eventid*: A-type address.
,ENTRYLENGTH=*elength*	Optional with TABLEDATA and not otherwise allowed. *elength*: A-type address.
,LOCBUFNAME=*bufname* ,LOCBUFADDR=*bufaddr*	Required choice with TABLEDATA and not otherwise allowed. *bufname*: Symbol up to eight characters long. *bufaddr*: A-type address.
,FILTERNAME=*pgmname* ,FILTERADDR=*pgmaddr*	Optional choice with TABLEDATA and not otherwise allowed. *pgmname*: Symbol up to eight characters long. *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* ,MODELADDR=*modeladdr*	Optional choice with EVENTDATA and not otherwise allowed. *modelname*: Symbol up to eight characters long. ,MODELADDR=*modeladdr*
,FORMATNAME=*pgmname* ,FORMATADDR=*pgmaddr*	Optional choice with EVENTDATA and not otherwise allowed. *pgmname*: Symbol up to eight characters long. *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 ,NOEXCEPTION	Optional choice with EVENTDATA and not otherwise allowed.

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.

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:

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)

© Copyright IBM Corp. 1988, 2002

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:

name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ITZXFILT.
ITZXFILT	
ƀ	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 70. 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 due to 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' '
```

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 page 687 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 page 693 for the syntax of this request.
- REQUEST=READBLOCK, which reads a selected log block in a log stream. See page 699 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 page 705 for the syntax of this request.
- REQUEST=END, which ends a browse session. See page 710 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, and IXGDELET.

Environment

The requirements for the caller are:

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
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.

© Copyright IBM Corp. 1988, 2002

Control parameters: All control parameters (except for ECB) must be in the primary address space with the following exceptions:

Any parameter which is explicitly ALET-qualified as allowed by the input parameters (for example, the area referenced by the BUFFER parameter when the BUFFALET parameter is also specified).

ECB, which should be addressable from the home address space. All storage areas must be in the same storage key as the caller, with the exception of the case where the parameter area is explicitly storage key qualified as allowed by the input parameters (for 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 parameter must be on a word boundary for multiple log block READCURSOR requests.
- When coding the ECB parameter, 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 storage of the home address space at the time the IXGBRWSE request is issued.

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.

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

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
b	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 GMT=NO	
VIEW=ACTIVE VIEW=ALL	**Default**: VIEW=ACTIVE
MODE=SYNC MODE=SYNCECB MODE=SYNCEXIT	**Default**: MODE=SYNC
,REQDATA=*reqdata*	*reqdata*: RS-type address or register (2) - (12).
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,DIAG=NO_DIAG ,DIAG=NO ,DIAG=YES	**Default**: DIAG=NO_DIAG
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default**: IMPLIED_VERSION
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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)	**Default**: MF=S

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 Greenwich mean time (GMT) 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 GMT or local time.

- GMT=YES: The time stamp specified on the SEARCH parameter is in GMT format.
- GMT=NO: The time stamp specified on the SEARCH parameter is local time.

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 definition in the LOGR 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.

The system where IXGBRWSE is issued must be IPLed at the OS/390 Release 3 level or above for the VIEW parameter to be recognized. If this parameter is specified on a pre-OS/390 release 3 level systems, it is processed as VIEW=ACTIVE.

,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 at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

,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.

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=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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
b	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).
,BUFFLEN=*bufflen*	*bufflen*: RS-type address or register (2) - (12).

,DIRECTION=OLDTOYOUNG
,DIRECTION=YOUNGTOOLD

,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).
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).

,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*
Specifies the name or address (using a register) of a required output field that contains the buffer into which the log block is read.

,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 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 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 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 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 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 Greenwich mean time stamp and the local time stamp associated with the requested log block are returned. The GMT 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 at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
b	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).
,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
,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 MODE=SYNCECB MODE=SYNCEXIT	**Default**: MODE=SYNC
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default**: IMPLIED_VERSION

,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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)	**Default**: MF=S

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*
Specifies the name or address (using a register) of a required output field that contains the buffer into which the log block is read.

,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 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 Greenwich mean time (GMT) 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 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 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 Greenwich mean time and local time stamps associated with the requested log block is returned. The GMT 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 at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
b	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 MODE=SYNCECB MODE=SYNCEXIT	**Default**: MODE=SYNC
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default**: IMPLIED_VERSION
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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)	**Default**: MF=S

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 definition in the LOGR 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.

The system where IXGBRWSE is issued must be IPLed at the OS/390 Release 3 level or above for the VIEW parameter to be recognized. If this parameter is specified on a pre-OS/390 release 3 level systems, it is processed as VIEW=ACTIVE.

,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 at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

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:

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
b	One or more blanks must follow IXGBRWSE.

REQUEST=END	
,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 MODE=SYNCECB MODE=SYNCEXIT	**Default**: MODE=SYNC
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default**: IMPLIED_VERSION
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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)	**Default**: MF=S

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.
- 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 at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

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: The return and reason codes are in the answer area mapped by IXGANSAA when the request completes.

The IXGCONN 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 71. 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 via an IXGDELET request. 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.
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 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 attempt to re-read the log data.

Table 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0405	**Equate Symbol**: IxgRsnCodeWarningLossOfData **Explanation**: Environment error. Returned for read requests. A log block has been returned, but there may be log blocks permanently missing between this log block and the one previously returned. This condition occurs when a system and coupling facility fail and 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.
04	xxxx0416	**Equate Symbol**: IxgRsnCodeWarningMultiblock **Explanation**: Environment error. Returned for READCURSOR requests with MULTIBLOCK=YES specified only. A log block has been returned, but at least one of the log blocks encountered a warning return code condition. **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.

Table 71. Return and Reason Codes for the IXGBRWSE 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	xxxx0803	**Equate Symbol**: IxgRsnCodeBadBuffer **Explanation**: Program error. The virtual storage area specified on the BUFFER 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 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 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 that the request was issued from the connector's address space.

Table 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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 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.

Table 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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	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**: 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 71. 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 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0845	**Equate Symbol**: IxgRsnCodeInvalidFunc **Explanation**: System error. The parameter list for this service contains an unrecognizable function code. The parameter list storage may have been overlaid. **Action**: Fix the problem and then re-issue the request.
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 via a prior IXGDELET request. **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.
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, 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 may 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.

Table 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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 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 attempt to re-issue the request.
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.

Table 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx084D	**Equate Symbol**: IxgRsnCodeLossOfDataEOR **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.
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.

Table 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 71. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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 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.

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
```

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 on using the system logger services and the IXGCONN request, see *z/OS MVS Programming: Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state with any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=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:	None.

© Copyright IBM Corp. 1988, 2002

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.

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 Security 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 access to RESOURCE(*log_stream_name*) in SAF class CLASS(LOGSTRM).
  - To connect to a log stream with an authorization level of WRITE, the caller must have alter access to RESOURCE(*log_stream_name*) in SAF class CLASS(LOGSTRM).

  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.
- An MVS image may connect to a maximum of 4096 log streams.

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

None.

Syntax

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

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGCONN.
IXGCONN	
b	One or more blanks must follow IXGCONN.

	Valid parameters (Required parameters are underlined.)
REQUEST=CONNECT REQUEST=DISCONNECT	All parameters are valid. STREAMTOKEN, ANSAREA, ANSLEN, USERDATA, RETCODE, RSNCODE, MF
,STREAMNAME=*streamname*	*streamname*: RS-type address or register (2) - (12).
,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 ,AUTH=WRITE	**Default**: AUTH=READ
,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 ,IMPORTCONNECT=YES	**Default**: IMPORTCONNECT=NO
,DIAG=NO_DIAG ,DIAG=NO ,DIAG=YES	**Default**: DIAG=NO_DIAG
,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 ,PLISTVER=1 ,PLISTVER=2	**Default**: IMPLIED_VERSION
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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)	**Default**: MF=S

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 policy, see the IXGINVNT macro for information on the syntax of log stream names in the LOGR 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.

,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 hexadecimal 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 LOGR 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 LOGR policy.

- If you are using an OS/390 Release 3 or higher LOGR 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 hexadecimal 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 hexadecimal 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 GMT 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 LOGR 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.

To use this parameter, the system must be at OS/390 Release 3 level or above.

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. The application must run in supervisor state, key 0 to use this 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 GMT 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.

,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 OS/390 Release 3 level or above 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 OS/390 Release 3 level or above 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 OS/390 Release 3 level or above 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 OS/390 Release 3 level or above 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.

Note: The return and reason codes are in the answer area mapped by IXGANSAA when the request completes.

The IXGCONN 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 72. Return and Reason Codes for the IXGCONN Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**Equate Symbol**: IxgRsnCodeOk **Explanation**: Request processed successfully.
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 re-build is in progress. **Action**: Listen to the ENF signal 48, which will indicate either that the log stream is available because the re-build completed successfully or that the log stream is not available because the re-build 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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
08	xxxx080B	**Equate Symbol**: IxgRsnCodeNoStream **Explanation**: Program error. The log stream name specified has not been defined in the LOGR policy. **Action**: Ensure that the required log stream name has been defined in the LOGR 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**: Define alter SAF authorization to the log stream or specify AUTH=READ.
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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0812	**Equate Symbol**: IxgRsnCodeLogStreamRecoveryFailed **Explanation**: Environment error. The log stream could not be recovered. The system issues message IXG211E 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**: Re-define the log stream in the LOGR policy and then re-issue 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**: Re-issue the request, specifying an answer area of the required size.
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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx081A	**Equate Symbol**: IxgRsnCodeMaxStreamConn **Explanation**: Environment error. The system has reached the limit for the maximum number of log streams that can be concurrently active. An MVS image may connect to a maximum of 4096 log streams concurrently. **Action**: Either plan your workload to either consolidate log streams or balance system activity so that fewer log streams are needed in a given time period.
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.
08	xxxx081F	**Equate Symbol**: IxgRsnCodeXcdsError **Explanation**: System error. System logger encountered an internal problem while processing the LOGR couple data set. **Action**: Contact the IBM Support Center. Provide the return and reason code and the contents of the answer area (ANSAREA field).

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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 LOGR 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 LOGR couple data set is available. 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 for the remainder of this IPL.
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.
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 define a resource manager for the log stream definition in the LOGR couple data set, or remove the RMNAME parameter from the request.
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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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	xxxx084F	**Equate Symbol**: IxgRsnCodeInvalidRMNameSpecified **Explanation**: Program error. The value for the RMNAME parameter on the connect request does not match the name of the resource manager defined in the LOGR couple data set for the log stream. **Action**: Either correct the RMNAME value on the connect request or correct the resource manager name in the log stream definition in the LOGR couple data set.
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.
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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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. 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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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 re-build 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. The connect request specified the RMNAME or IMPORTCONNECT parameter. The request failed because the active primary LOGR couple data set must be at OS/390 Release 3 or above to support these parameters. **Action**: Either retry the request without the RMNAME or IMPORTCONNECT parameters or reformat the LOGR couple data set at OS/390 Release 3 or above level.
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.

Table 72. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx08E2	**Equate Symbol**: IxgRsncodeDasdOnlyConnected **Explanation**: Environment error System logger rejected an attempt to connect to a DASD-only log stream because the log stream is already connected to by another log stream in the sysplex. Only one system at a time can connect to a DASD-only log stream. **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.
08	xxxx08E3	**Equate Symbol**: IXGRSNCODELOGSTREAMNOTSUPPORTED **Explanation**: Environment error. A request to connect to a log stream was rejected on this system because the system release level does not support a DASD-only log stream. The system must be at OS/390 Release 3 or higher to connect to DASD-only log stream. **Action**: If you must connect to a DASD-only log stream, make sure you do one of the following: • Connect from a system that is OS/390 Release 3 or higher. • Update the log stream definition in the LOGR policy to a coupling facility one by specifying a structure name on the definition. (This can only be done on a system that is OS/390 R3 or higher.) • Delete the log stream definition from the LOGR policy, and redefine it as a coupling facility log stream with an associated structure name. Then a system at a level below OS/390 R3 can connect to the log stream. If the log stream was never connected to by any system, the delete request can be done from a system of any level. Otherwise, the delete request must be done from an OS/390 R3 or higher system.
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
```

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, and IXGINVNT.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state with any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=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:	All control parameters (except for ECB) must be in the primary address space. ECB should be addressable from home address space.

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.

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.

© Copyright IBM Corp. 1988, 2002

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:

---

name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede IXGDELET.
IXGDELET	
ƀ	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 ,FORCE=YES	**Default**: FORCE=NO
,FORCEINFO=NO	**Default**: FORCEINFO=NO
,OBLOCKID=*oblockid*	*oblockid*: RS-type address or register (2) - (12).
MODE=SYNC MODE=ASYNCNORESPONSE MODE=SYNCECB MODE=SYNCEXIT	**Default**: MODE=SYNC
,REQDATA=*reqdata*	*reqdata*: RS-type address or register (2) - (12).
,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,DIAG=NO_DIAG ,DIAG=NO ,DIAG=YES	**Default**: DIAG=NO_DIAG
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=0 ,PLISTVER=1	**Default**: IMPLIED_VERSION
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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)	**Default:** MF=S

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.

To use this parameter, the system where the application is running must be IPLed at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

- 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 at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

,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: The return and reason codes will also be put into the answer area mapped by IXGANSAA when the request completes.

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 73. Return and Reason Codes for the IXGDELET 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	xxxx040B	**Equate Symbol**: IxgRsnCodeRMNotConnected **Explanation**: Program or environment error. The log stream is identified as being a source log stream managed by a resource manager (RMNAME is specified in the LOGR couple data set). 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 a resource manager managed system if the resource manager is connected to the log stream. **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.

Table 73. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 73. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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 LOGR couple data set). 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 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.
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	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.

Table 73. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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	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.
08	xxxx081F	**Equate Symbol**: IxgRsnCodeXcdsError **Explanation**: System error. System logger encountered an internal problem while processing the LOGR 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.

Table 73. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 73. Return and Reason Codes for the IXGDELET 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.
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.
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 73. 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.

Table 73. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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
```

IXGIMPRT — Import Log Blocks

Description

The IXGIMPRT macro allows a program to import a copy of a log block from one log stream to another, specifying a log block identifier and time stamp to be assigned to the log block.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state. Any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=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 may be held.
Control parameters:	None.

Programming Requirements

- Before issuing this request, the caller must have a valid connection to the log stream. The connection must be issued with AUTH=WRITE and IMPORTCONNECT=YES parameters specified.
- 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.

Restrictions

All storage areas specified must be in the same storage key as the caller. Storage areas that are not ALET qualified must exist in the caller's primary address space.

Input Register Information

Before issuing the IXGIMPRT 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

© Copyright IBM Corp. 1988, 2002

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 IXGIMPRT macro is written as follows:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGIMPRT.
IXGIMPRT	
b	One or more blanks must follow IXGIMPRT.

STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or address in register (2) - (12).
,BUFFER=*buffer*	*buffer*: RS-type address or address in register (2) - (12).
,BLOCKLEN=*blocklen*	*blocklen*: RS-type address or address in register (2) - (12).
,BLOCKID=*blockid*	*blockid*: RS-type address or address in register (2) - (12).
,GMT_TIMESTAMP=*gmt_timestamp*	*gmt_timestamp*: RS-type address or address in register (2) - (12).
,LOCALTIME=*localtime*	*localtime*: 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).
,BUFFALET=*buffalet* ,BUFFALET=**0,**	*buffalet*: RS-type address or address in register (2) - (12). **Default**: BUFFALET=0,
,BUFFKEY=*buffkey* ,BUFFKEY=*****	*buffkey*: RS-type address or address in register (2) - (12). **Default**: BUFFKEY=*
,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	**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=(E,*list addr*,NOCHECK) ,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 IXGIMPRT macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

STREAMTOKEN=*streamtoken*
A required input parameter that specifies the log stream token that was returned by the IXGCONN service.

To code: Specify the RS-type address, or address in register (2)-(12), of a 16-character field.

,BUFFER=*buffer*
A required input parameter that specifies the buffer from which the log stream block is to be written.

The buffer can be ALET qualified. If a buffer is ALET qualified, the ALET must index a valid entry on the task's dispatchable unit access list (DUAL).

To code: Specify the RS-type address, or address in register (2)-(12), of a character field.

,BLOCKLEN=*blocklen*
A required input parameter that specifies the length of the log block to be written. The maximum block length is 65,536.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,BLOCKID=*blockid*
A required input parameter that specifies the block id to be assigned to the log block being written. The block identifier specified must be greater than any previous block identifier in the log stream.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,GMT_TIMESTAMP=*gmt_timestamp*
A required input parameter that specifies the 8-byte GMT time stamp to be associated with the log block being written. The timestamp specified must be greater than any previous timestamp in the log stream. The timestamp must be in STCK format.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,LOCALTIME=*localtime*
A required input parameter that specifies the 8-byte local time stamp to be associated with the log block being imported. The timestamp must be in STCK format.

To code: Specify the RS-type address, or address in register (2)-(12), of an 8-character field.

,ANSAREA=*ansarea*
A required output parameter of a virtual storage area, called the answer area, in which service response information will be placed. The format of the answer area is described by the **IXGANSAA** mapping macro.

To code: Specify the RS-type address, or address in register (2)-(12), of a field.

,ANSLEN=*anslen*
A required input parameter that specifies the answer area length. The length of the answer area must be at least as large as the length of IXGANSAA.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,BUFFALET=*buffalet*
,BUFFALET=0,
An optional input parameter that specifies the ALET to be used to access the storage specified by the **BUFFER** keyword. The default is 0, which means that the buffer resides in the caller's primary address space.

To code: Specify the RS-type address, or address in register (2)-(12), of a fullword field.

,BUFFKEY=*buffkey*
,BUFFKEY=*
An optional input parameter that specifies the KEY associated with the storage specified by the **BUFFER** keyword. Callers invoking this service must specify the execution key (PSW key). The default is * execution key (PSW key), which means that if buffer key is not specified, the execution key (PSW key) of the caller will be used as the default.

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

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.

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)
,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

Abend 1C5 **Ixg_Abend_Code -** See *z/OS MVS System Codes* for more information on this abend.

Return and Reason Codes

When the IXGIMPRT 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 reason code.

The IXGCONN 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. 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 74. Return and Reason Codes for the IXGIMPRT Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**IxgRsnCodeOk -** **Explanation**: Request processed successfully.

Table 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0405	**IxgRsnCodeWarningLossOfData -** **Explanation**: Environment error. The request was successful however the log stream has previously lost log blocks. This condition occurs when a system and coupling facility fail and 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.
04	xxxx0407	**IxgRsnCodeConnPossibleLossOfData -** **Explanation**: Environment error. The request was successful, but there may be log blocks permanently in the log stream. 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	**IxgRsnCodeDsDirectoryFullWarning -** **Explanation**: Environment error. The request was successful, but the log stream's 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 IXGIMPRT 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 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx040A	**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 (STG_DUPLEX=YES, DUPLEXMODE=UNCOND). **Action**: 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.
08	xxxx0801	**IxgRsnCodeBadParmlist -** **Explanation**: Program error. The parameter list is invalid. Either the parameter list storage is inaccessible, or an invalid version of the macro was used. **Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request, and that the macro version is correct. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx0802	**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	**IxgRsnCodeBadBuffer -** **Explanation**: Program error. The virtual storage area specified on the BUFFER parameter is not addressable. **Action**: Ensure that the storage area specified on the BUFFER 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	**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 connectors address space. **Action**: Do one of the following: • Make sure that the stream token specified is valid. • Ensure that IXGIMPRT requests were issued from the connectors address space.

Table 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0809	**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 LOGR policy 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	**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	**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	**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	**IxgRsnCodeBadAnsIen -** **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.
08	xxxx0817	**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 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0819	**IxgRsnCodeSRBMode -** **Explanation**: Program error. The calling program is in SRB mode, but task mode is required for this system logger service. **Action**: Make sure your program is in task mode.
08	xxxx082D	**IxgRsnCodeExpiredStmToken -** **Explanation**: Environment error. The stream token is no longer valid because the connector has been disconnected. **Action**: Re-connect to the logstream before issuing any functional requests.
08	xxxx083F	**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
08	xxxx0840	**IxgRsnCodeBadVersion -** **Explanation**: Environment error. The parameter list passed to the service routine has an incorrect version indicator. **Action**: Make sure that the level of MVS executing the request and the macro library used to compile the invoking routine are compatible.
08	xxxx0841	**IxgRsnCodeBadBufferAlet -** **Explanation**: Program error. The buffer ALET specified is not zero and does not represent a valid entry on the callers 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	**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).

Table 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx085C	**IxgRsnCodeDsDirectoryFull -** **Explanation**: Environment error. The coupling facility structure space allocated for this log stream is full, or the staging data set space is full. Attempts to offload the coupling facility data to DASD 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. The sytem issues related messages IXG257I and ISG301I. **Action**: You must make more log stream data set directory space available. See *z/OS MVS Setting Up a Sysplex* for more information.
08	xxxx085D	**Equate Symbol**: IxgRsnCodeWowError **Explanation**: Environment error. The coupling facility structure space allocated for this log stream is full, or the staging data set space is full. Attempts to offload log data to DASD failed because of severe errors. No further write requests can be processed. **Action**: Wait for the ENF signal that indicates the log stream has been sucessfully offloaded, or disconnect from this log stream, and connect to another log stream. Check log for message IXG301I to determine the cause of the error. If error was related to your installation, correct the error. Otherwise, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
08	xxxx0860	**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 and then reissue the request.
08	xxxx0861	**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. Reissue the request. • The re-build failed and the log stream is not available.

Table 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0862	**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. Reissue the request. • The re-build failed and the log stream is not available.
08	xxxx0863	**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. Reissue the request. • The re-build failed and the log stream is not available.
08	xxxx0864	**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. Reissue the request. • The re-build failed and the log stream is not available. # The log stream has been disconnected from this system.
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, reissue the request.
08	xxxx0867	**Equate Symbol**: IxgRsnCodeLocalBufferFull **Explanation**: Environment error. The available local buffer space for the system logger address space is full. No further requests can be processed until the log data in the local storage buffer is offloaded to DASD log data sets. Note that this reason code applies only to a IXGWRITE or IXGIMPRT request issued against a DASD-only log stream. **Action**: Listen for the ENF signal 48 indicating that the DASD-only log stream is available again after the data has been offloaded to DASD log data sets. Then reissue the request.

Table 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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 ENG signal 48 which will indicate that the log stream is available after formatting process is finished. Then, reissue the request.
08	xxxx0890	**IxgRsnCodeAddrSpaceNotAvail -** **Explanation**: System error. The system logger address space failed and is not available. **Action**: Do not issue system logger requests.
08	xxxx0891	**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. Once it's available, re-connect to the log stream, then 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.
08	xxxx08D7	**IxgRsnCodeRequestNotAllowed -** **Explanation**: Program error. The caller attempted to issue an import request while a write connection (IXGCONN AUTH=WRITE,IMPORTCONNECT=NO) was active. **Action**: Issue the correct type of request based on the import status of your connection.
08	xxxx08D9	**IxgRsnCodeBadImportBlockId -** **Explanation**: Program error. The blockid specified on the import request was either less than the blockid expected or less than the size the control information system logger adds to each log block. You can use IXGQUERY service to ascertain the size of control information for a log block. IXGQUERY returns the control information size for a log stream in the QBUF_Control_Info_Size field in the query buffer. IXGQUERY also returns the block identifier of the last successfully written log block. **Action**: Specify a valid value for the block id and reissue the import request.
08	xxxx08DA	**IxgRsnCodeBadImportTimeStamp -** **Explanation**: Program error. The GMT timestamp specified on the import request was not greater than or equal to the GMT time stamp assigned to the last log block successfully imported. **Action**: Specify a valid value for GMT_TimeStamp and reissue the request. You can obtain the GMT timestamp of the last successfully written log block using the IXGQUERY service.

Table 74. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx08DB	**IxgRsnCodeImportNoSrbMode -** **Explanation**: Program error. IXGIMPRT requests can only be issued in task mode. **Action**: Issue the IXGIMPRT request while executing in task mode.
08	xxxx08DC	**IxgRsnCodeImportInProgress -** **Explanation**: Program error. Only one import operation for a given log stream can be in progress at any instance in time. The problem may be due to a task initiating an import request before a previously initiated import to the log stream has completed. **Action**: Wait for the currently executing import operation to complete before initiating a subsequent import operation.
0C	xxxx0000	**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

Issue IXGIMPRT to import a log block to a back up log stream.

```
* R6     Read buffer address
         IXGIMPRT                                                      X
               STREAMTOKEN=OTOKEN,                                     X
               BUFFER=(R6),                                            X
               BLOCKLEN=DATALEN,                                       X
               BLOCKID=RBLKID,                                         X
               GMT_TIMESTAMP=GMTTIME,                                  X
               LOCALTIME=LOCTIME,                                      X
               ANSAREA=XANSAREA,                                       X
               ANSLEN=XANSLEN,                                         X
               RSNCODE=RSCODE
R6       EQU   6
OTOKEN   DS    CL16                    Output Stream token
DATALEN  DS    F                       Returned data length
RBLKID   DS    CL8                     Returned block identifier
GMTTIME  DS    CL8                     GMT
LOCTIME  DS    CL8                     Local Time
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
```

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, and IXGDELET.

Environment

The requirements for the caller are:

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-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 (except for ECB) must be in the primary address space. ECB must be addressable from home address space.

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.

© Copyright IBM Corp. 1988, 2002

- 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.

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.

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:

---

name	*name*: symbol. Begin *name* in column 1.
b	One or more blanks must precede IXGWRITE.

IXGWRITE

b	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).
,BLOCKLEN=*blocklen*	*blocklen*: RS-type address or register (2) - (12).
,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=*	*buffkey*: RS-type address or register (2) - (12). **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 MODE=ASYNCNORESPONSE MODE=SYNCECB MODE=SYNCEXIT	**Default**: MODE=SYNC
,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 ,PLISTVER=*plistver* ,PLISTVER=0 ,PLISTVER=1	**Default**: IMPLIED_VERSION
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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)	**Default**: MF=S

,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*
Specifies the field name (or address in a register) of the data to be written to the log.

,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 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 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.

  To use this parameter, the system where the application is running must be IPLed at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.
- 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.

  To use this parameter, the system where the application is running must be IPLed at OS/390 Release 3 level or above. If you specify this request on a pre-OS/390 Release 3 level system, the request is processed as a MODE=SYNC request.

,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: The return and reason codes will be put into the answer area mapped by IXGANSAA only when system logger completes asynchronous processing of the request.

The IXGCONN macro provides equate symbols for the return and reason codes. The equate symbols associated with each hexadecimal return code are as follows:

00 IXGRSNCODEOK
04 IXGRSNCODEWARNING
08 IXGRETCODEERROR
0C IXGRETCODECOMPERROR

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 75. 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**: 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	xxxx0405	**Equate Symbol**: IxgRsnCodeWarningLossOfData **Explanation**: Environment error. The request was successful however the log stream has previously lost log blocks. This condition occurs when a system and coupling facility fail and 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.
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.

Table 75. Return and Reason Codes for the IXGWRITE Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
04	xxxx040A	**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 (STG_DUPLEX=YES, DUPLEXMODE=UNCOND). **Action**: 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.
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 75. Return and Reason Codes for the IXGWRITE 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 parameter is not addressable. **Action**: Ensure that the storage area specified on the BUFFER 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.
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 LOGR policy 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.

Table 75. Return and Reason Codes for the IXGWRITE Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 75. Return and Reason Codes for the IXGWRITE Macro (continued)

Return Code	Reason Code	Meaning and Action
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	xxxx085C	**Equate Symbol**: IxgRsnCodeDsDirectoryFull **Explanation**: Program error. The coupling facility structure space allocated for this log stream is full, or the staging data set space is full. Attempts to offload the coupling facility data to DASD 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. The sytem issues related messages XG257I and ISG301I. **Action**: You must make more log stream data set directory space available. See *z/OS MVS Setting Up a Sysplex* for more information.

Table 75. Return and Reason Codes for the IXGWRITE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx085D	**Equate Symbol**: IxgRsnCodeWowError **Explanation**: Environment error. The coupling facility structure space allocated for this log stream is full, or the staging data set space if full. Attempts to offload the interim storage data to DASD failed because of severe errors. No further write requests can be processed. **Action**: Wait for the ENF signal that the log stream has been successfully offloaded, or disconnect from this log stream and connect to another log stream. Check log for message message IXG301I to determine the cause of the error. If error was related to the your installation, correct the error. Otherwise, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
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.

Table 75. Return and Reason Codes for the IXGWRITE 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.
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. The available local buffer space for the system logger address space is full. No further requests can be processed until the log data in the local storage buffer is offloaded to DASD log data sets. Note that this reason code applies only to a IXGWRITE or IXGIMPRT request issued against a DASD-only log stream. **Action**: Listen for the ENF signal 48 indicating that the DASD-only log stream is available again after the data has been offloaded to DASD log data sets. Then re-issue the request.
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 75. 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. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen-readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size

Using assistive technologies

Assistive technology products, such as screen-readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using it to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to *z/OS TSO/E Primer*, *z/OS TSO/E User's Guide*, and *z/OS ISPF User's Guide Volume I* for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

© Copyright IBM Corp. 1988, 2002

Notices

This information was developed for products and services offered in the USA.

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
USA

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

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

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

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

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

© Copyright IBM Corp. 1988, 2002

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
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This book is intended to help the customer to code macros that are available to authorized assembler language programs. This book documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/OS.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- AFP
- CICS
- DFSMS
- DFSMSdfp
- ESA/390
- ESCON
- Hiperspace
- IBM
- IBMLink
- MVS/ESA
- MVS/SP
- MVS/XA
- OS/390
- RACF
- Resource Link
- SecureWay
- SP
- SP1
- SP2
- z/Architecture
- z/OS
- z/OS.e

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

Other company, product and service names may be the trademarks or service marks of others.

Index

Numerics

31-bit addressing mode
  macros requiring expansion
    ESTAE macro 83

A

abend
  interrupting scheduled 83
accessibility 799
addressing mode and the services 2
ALET qualification
  of parameters 3
AR () mode
  description 3
ASC (address space control) mode
  defining 3

C

callable service
  coding 15
captured UCB
  obtaining actual address 551
coding the callable services 15
coding the macros 12
component trace format table
  generating 677
continuation line 15

D

data sharing with IARVSERV macro 235
device measurement block index
  obtaining 533
disability 799
documents, licensed xii

E

ENFREQ macro 25
ENQ macro 55
entry table
  connecting 99
  creating 105
  destroying 119
  disconnecting 123
entry table descriptor
  creating 109
ESPIE macro 73
ESTAE and ESTAEX macros 83
ETCON macro 99
ETCRE macro 105
ETDEF macro 109
ETDES macro 119
ETDIS macro 123
event
  notification of occurrence 25
  waiting for completion 127
EVENTS macro 127
EXTRACT macro 133

F

FESTAE macro 139
FRACHECK macro (for RACF Release 1.8.1 or earlier) 143
FREEMAIN macro 145

G

GETDSAB macro 157
GETMAIN macro 165
global serialization queue
  extracting information 179
GQSCAN macro 179
GTRACE macro 191
  DATA function 195
  QUERY function 193
  TEST function 191

H

hiperspace
  reading to 201
  writing from 201
HSPSERV macro 201

I

I/O configuration token
  obtaining 533
IARR2V macro 219
IARSUBSP macro 225
IARV64 macro 247
IARVSERV macro
  data sharing 235
IAZXCTKN macro 295
IAZXJSAB macro 297
IEAARR macro 301
IEAFP macro 305
IEALSQRY macro 309
IEAMRMF3 macro 313
IEAMSCHD macro 321
IEANTCR callable service 335
IEANTDL callable service 341
IEANTRT callable service 345
IEARBUP RB Update Service 351
IEAVAPE callable service 379
IEAVDPE callable service 383
IEAVPSE callable service 387
IEAVRLS callable service 393
IEAVRPI callable service 399

© Copyright IBM Corp. 1988, 2002

IEAVTPE callable service 405
IEAVXFR callable service 409
IEECMDS macro 415
IEEQEMCS macro 425
IEEVARYD macro 441
  comparison to MGCRE macro 441
IEFPPSCN macro 453
IEFQMREQ macro 461
IEFSSI macro 465
IEFSSVT macro 499
IEFSSVTI macro 519
IOCINFO macro 533
IOS (input/output supervisor)
  building control unit entry 599
  ENQ 607
  obtain switch information 635
  obtaining information 585, 591, 613, 629
IOSADMF macro 539
IOSCAPF macro 551
IOSCAPU macro 553
IOSCDR macro 571
IOSCHPD macro 585
IOSCMXA macro 591
IOSCMXR macro 597
IOSCUMOD macro 599
IOSDCXR macro 603
IOSENQ macro 607
IOSINFO macro 613
IOSLOOK macro 617
IOSPTHV macro 619
IOSUPFA macro 629
IOSUPFR macro 633
IOSWITCH macro 635
IRDFSD macro 641
IRDFSDU macro 647
ISGLCRT callable service 651
  syntax 651
ISGLOBT callable service 657
  syntax 657
ISGLPBA callable service
  syntax 673
ISGLPRG callable service 669
  syntax 669
ISGLREL callable service 663
  syntax 663
ITTFMTB macro 677
ITZXFILT macro 681
IXGBRWSE macro 685
IXGCONN macro 729
IXGDELET macro 751
IXGIMPRT macro 767
IXGWRITE macro 781

J

job
  obtaining information about current 297

K

keyboard 799

L

latch
  obtaining 657
  purging 669
  purging a group 673
  purpose 657, 663, 669, 673
  releasing 663
latch manager services
  ISGLCRT callable service
    syntax 651
  ISGLOBT callable service
    syntax 657
  ISGLPBA callable service
    syntax 673
  ISGLPRG callable service
    syntax 669
  ISGLREL callable service
    syntax 663
latch set
  creating 651
  purpose 651
licensed documents xii
linkage stack
  query macro 309
LookAt message retrieval tool xii

M

macro
  coding 12
  forms 11
  level
    selecting 1
  sample 13
  selecting level 1
  user parameter, passing 4
  X-macros
    using 10
message retrieval tool, LookAt xii

N

Notices 801

S

serially reusable resource
  requesting control 55
service
  ALET qualification 3
  summary 16
services
  addressing mode 2
  ASC mode
    defining 3
  using 1
sharing storage with IARVSERV macro 235
shortcut keys 799
subchannel number
  obtaining for a UCB 613

SWA manager
invoking in move mode 461

T

TCB (task control block)
extracting information 133

U

unit control block
capturing and releasing 553
locating 617
obtaining address of common extension segment 597
obtaining address of device class extension 603
obtaining address of prefix extension 633
user parameter
passing 4

V

virtual storage
allocating 145, 165
Virtual storage
sharing with IARVSERV macro 235

X

X-macros
using 10

Readers' Comments — We'd Like to Hear from You

z/OS
MVS Programming: Authorized
Assembler Services Reference, Volume 2
(ENFREQ-IXGWRITE)

Publication No. SA22-7610-03

Overall, how satisfied are you with the information in this book?

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Overall satisfaction	☐	☐	☐	☐	☐

How satisfied are you that the information in this book is:

	Very Satisfied	Satisfied	Neutral	Dissatisfied	Very Dissatisfied
Accurate	☐	☐	☐	☐	☐
Complete	☐	☐	☐	☐	☐
Easy to find	☐	☐	☐	☐	☐
Easy to understand	☐	☐	☐	☐	☐
Well organized	☐	☐	☐	☐	☐
Applicable to your tasks	☐	☐	☐	☐	☐

Please tell us how we can improve this book:

Thank you for your responses. May we contact you? ☐ Yes ☐ No

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

Name _____ Address _____

Company or Organization _____

Phone No. _____

Readers' Comments — We'd Like to Hear from You
SA22-7610-03

Cut or Fold
Along Line

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation
Department 55JA, Mail Station P384
2455 South Road
Poughkeepsie, NY
12601-5400

Fold and Tape | **Please do not staple** | Fold and Tape

SA22-7610-03

Cut or Fold
Along Line

Program Number: 5694-A01, 5655-G52

Printed in U.S.A.

SA22-7610-03