z/OS

MVS Programming: Assembler Services Reference, Volume 2 (IARR2V-XCTLX)

SA22-7607-03

z/OS

MVS Programming: Assembler Services Reference, Volume 2 (IARR2V-XCTLX)

SA22-7607-03

Note

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 803.

Fourth Edition, September 2002

This is a revision of SA22–7607–02.

This edition applies to Version 1 Release 4 of z/OS (5694-A01), 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

IARR2V — Convert a Central Storage Address to a Virtual Storage Address 23

IARVSERV — Request to Share Virtual Storage 29

IARV64 — 64–Bit Virtual Storage Allocation 41

IDENTIFY — Add an Entry Name 73

IEAARR — Establish an Associated Recovery Routine (ARR) 77

IEABRC — Relative Branch Macro 81

IEAFP — Floating Point Services 83

IEAINTKN — Build Incident Token 87

IEALSQRY — Linkage Stack Query 89

IEANTCR — Create a Name/Token Pair 93

IEANTDL — Delete a Name/Token Pair 99

IEANTRT — Retrieve the Token from a Name/Token Pair 103

IEATDUMP — Transaction Dump Request 109

IEAVAPE — Allocate_Pause_Element 127

IEAVDPE — Deallocate_Pause_Element 131

IEAVPSE — Pause Service 135

IEAVRLS — Release 141

IEAVRPI — Retrieve_Pause_Element_Information Service 145

IEAVTPE — Test_Pause_Element Service 151

IEAVXFR — Transfer Service 155

IEFDDSRV — Receive Device Information For an Allocation Request 161

IEFPRMLB — Logical Parmlib Support 169

© Copyright IBM Corp. 1988, 2002

IEFSSI — Dynamically Query a Subsystem 197

IOCINFO — Obtain MVS I/O Configuration Information 205

IOSCHPD — IOS CHPID Description Service 211

IOSCUMOD — IOS Control Unit Entry Build Service 217

ITTUINIT — Activate external CTRACE recording 221

ITTUTERM — End external CTRACE recording 225

ITTUWRIT — Queue a group of CTRACE entries 229

ITZEVENT — Transaction Trace EVENT Record 233

ITZQUERY — Transaction Trace Query 243

IXGBRWSE — Browse/Read a Log Stream 249

IXGCONN — Connect/Disconnect to Log Stream 293

IXGDELET — Deleting Log Data from a Log Stream 315

IXGIMPRT — Import Log Blocks 331

IXGINVNT — Managing the LOGR Inventory Couple Data Set 345

IXGOFFLD — Initiate Offload to DASD Log Data Sets 401

IXGQUERY — Query a Log Stream for Information 411

IXGUPDAT — Update Log Stream Control Information 421

IXGWRITE — Write Log Data to a Log Stream 431

LINK and LINKX — Pass Control to a Program in Another Load Module 447

LOAD — Bring a Load Module into Virtual Storage 457

LSEXPAND — Expand a Linkage Stack to a Specified Size 465

PGLOAD — Load Virtual Storage Areas into Central Storage 469

PGOUT — Page Out Virtual Storage Areas from Central Storage 473

PGRLSE — Release Virtual Storage Contents 477

PGSER — Page Services 481

POST — Signal Event Completion 487

QRYLANG — Determine Languages Available for Message Translation 491

REFPAT — Define and End a Reference Pattern 497

RESERVE — Reserve a Device (Shared DASD) 505

RETURN — Return Control 515

SAVE — Save Register Contents 517

SETRP — Set Return Parameters 519

SNAP and SNAPX — Dump Virtual Storage and Continue 525

SPIE — Specify Program Interruption Exit 539

SPLEVEL — Set Macro Level 545

STAE — Specify Task Abnormal Exit 549

STATUS — Start and Stop a Subtask 555

STCKCONV — Store Clock Conversion Routine 561

STCKSYNC — Store Clock Synchronous Service 567

STIMER — Set Interval Timer 571

STIMERM — Set, Test, Cancel Multiple Interval Timer 577

STORAGE — Obtain and Release Storage 591

SYMRBLD — Building a Symptom Record 605

SYMREC — Process a Symptom Record 623

SYNCH and SYNCHX — Take a Synchronous Exit to a Processing Program 631

SYSEVENT — System Event 639

SYSSTATE — Identify System State 641

TCBTOKEN — Request or Translate the TTOKEN 645

TESTART — Tests the Validity of ALETs 651

TIME — Obtain Time and Date 655

TIMEUSED — Obtain Accumulated CPU or Vector Time 665

TRANMSG — Translate Messages 669

TTIMER — Test Interval Timer 683

UCBDEVN — Return EBCDIC Device Number for a UCB 687

UCBINFO — Return Information from a UCB 691

UCBSCAN — Scan UCBs 727

UPDTMPB — Update a Message Parameter Block for Substitution Data 739

VRADATA — Update Variable Recording Area Data 745

WAIT — Wait for One or More Events 751

WTL — Write To Log 757

WTO — Write to Operator 765

WTOR — Write to Operator with Reply 779

XCTL and XCTLX — Pass Control to a Program in Another Load Module 791

Appendix. Accessibility 801

Notices 803

Index 807

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 RESERVE 509

© 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. Return and Reason Codes for the IARR2V Macro . . . 26
5. Return and Reason Codes for the IARVSERV Macro . . . 34
6. Return and Reason Codes for the IARV64 Macro . . . 70
7. Return and Reason Codes for the IEAFP Macro . . . 85
8. Return Codes for IEALSQRY . . . 91
9. Return and Reason Codes for the IEATDUMP Macro . . . 119
10. Return and Reason Codes for the IEFDDSRV Macro . . . 164
11. Return and Reason Codes for the IEFPRMLB Macro . . . 178
12. Return and Reason Codes for the IEFSSI Macro . . . 202
13. Return and Reason Codes for the IOSCHPD Macro . . . 215
14. Return and Reason Codes for the ITZEVENT Macro . . . 240
15. Return and Reason Codes for the ITZQUERY Macro . . . 247
16. Return and Reason Codes for the IXGBRWSE Macro . . . 277
17. Return and Reason Codes for the IXGCONN Macro . . . 301
18. Return and Reason Codes for the IXGDELET Macro . . . 321
19. Return and Reason Codes for the IXGIMPRT Macro . . . 336
20. Return and Reason Codes for the IXGINVNT Macro . . . 383
21. Return and Reason Codes for the IXGOFFLD Macro . . . 406
22. Return and Reason Codes for the IXGQUERY Macro . . . 416
23. Return and Reason Codes for the IXGUPDAT Macro . . . 426
24. Return and Reason Codes for the IXGWRITE Macro . . . 437
25. Return and Reason Codes for the LSEXPAND Macro . . . 467
26. Return Codes for the RESERVE Macro with the RET=TEST Parameter . . . 509
27. Return Codes for the RESERVE Macro with the RET=USE Parameter . . . 510
28. Return Codes for the RESERVE Macro with the RET=HAVE Parameter . . . 511
29. Return Codes for the STATUS Macro . . . 557
30. Return Codes for the STCKCONV Macro . . . 564
31. Return Codes for the STCKSYNC Macro . . . 569
32. Return Codes for the STIMERM Macro . . . 584
33. Return Codes for STORAGE OBTAIN . . . 599
34. Return Codes for the STORAGE RELEASE . . . 602
35. Valid SDB Key Names and Literals . . . 611
36. Valid Section 5 Key Names and Literals . . . 617
37. Return Codes for the TCBTOKEN Macro . . . 647
38. Return Codes for the TESTART Macro . . . 653
39. Return Codes for the TIME Macro . . . 659
40. Return and Reason Codes for the TIMEUSED Macro . . . 667
41. Return Codes for the TTIMER Macro . . . 685
42. MCSFLAG Flag Names for WTO Macro . . . 769
43. MCSFLAG Flag Names for WTOR Macro . . . 782

© Copyright IBM Corp. 1988, 2002

About this document

This document describes some of the macros (or macro instructions) that the system provides. The macros described in this document are available to any assembler language program.

Programmers who code in assembler language can use these macros 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.

Who should use this document

This document is for any programmer who is coding an assembler language program. However, if the program resides on an APF-authorized library, runs in supervisor state or with system key 0-7, or if it performs functions that are more system than application-oriented, the programmer should also refer to the following documents:

- *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*

Programmers using this document should have a knowledge of the computer, as described in *Principles of Operation*, as well as a 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*.

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

© Copyright IBM Corp. 1988, 2002

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:

`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.

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

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 SA22-7607-03
z/OS Version 1 Release 4

The document contains information previously presented in *z/OS MVS Programming: Assembler Services Reference IAR-XCT*, SA22-7607-02, which supports z/OS Version 1 Release 3.

Information is added to indicate this document 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 SA22-7607-02
z/OS Version 1 Release 3

The document contains information previously presented in *z/OS MVS Programming: Assembler Services Reference IAR-XCT*, SA22-7607-01, which supports z/OS Version 1 Release 2.

New information

- The following macros support 64-bit addressing mode (AMODE): IARVSERV, IDENTIFY, LINKX, LOAD, PGSER, POST, RESERVE, STIMER, STIMERM, TIME, TIMEUSED, VRADATA, WAIT, WTO, WTOR, XCTLX.
- 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 SA22-7607-01
z/OS Version 1 Release 2

The document contains information previously presented in *z/OS MVS Programming: Assembler Services Reference IAR-XCT*, SA22-7607-00, which supports z/OS Version 1 Release 1.

New information

The following new service is added:

- IARV64 – 64–bit Virtual Storage Allocation

Changed information

© Copyright IBM Corp. 1988, 2002

- “Using the Services” on page 1 has been changed to reflect 64–bit Virtual Storage addressing.
- IARR2V — Convert a Central Storage Address to a Virtual Storage Address has been changed to include the new VSA64 keyword in support of 64–bit Virtual Storage.
- IXGINVNT — Managing the LOGR Inventory Couple Data Set has been changed to include the new LOGGERDUPLEX keyword in support of the provision of the capability of automatically duplexing coupling facility.
- LOAD — Bring a Load Module into Virtual Storage is changed to provide support for APAR OW18167.
- SYSSTATE — Identify System State is changed to add the new parameters, ARCHLVL and AMODE64 in support of z/Architecture.

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

Summary of changes
for SA22-7607-00
z/OS Version 1 Release 1

The document contains information also presented in *OS/390 MVS Programming: Assembler Services Reference*.

New information

The following new service is added:

- IOSCUMOD — IOS Control Unit Entry Build Service

Changed information

The following service is changed:

- IOSCHPD — IOS CHPID Description Service

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: Assembler Services 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.

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 LINK/LINKX XCTL/XCTLX	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, LINK/LINKX, or XCTL/XCTLX macros, 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
```

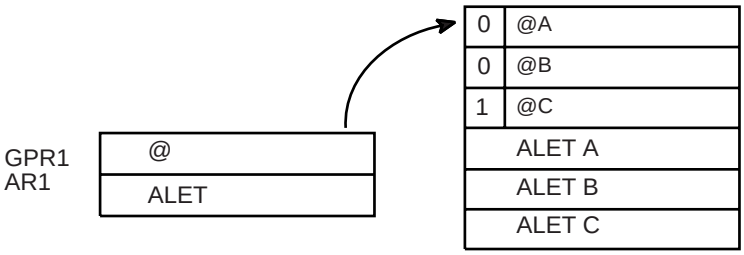

Figure 1. Sample User Parameter List for Callers in AR Mode

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

Characteristic	Parameter on SYSSTATE	Global symbol
Primary or AR ASC mode	ASCENV=P or AR	&SYSASCE
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: 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 X-macros described in this book are:
- ATTACHX
- ESTAEX
- LINKX
- SNAPX
- SYNCHX
- XCTLX

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, in which case, 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.

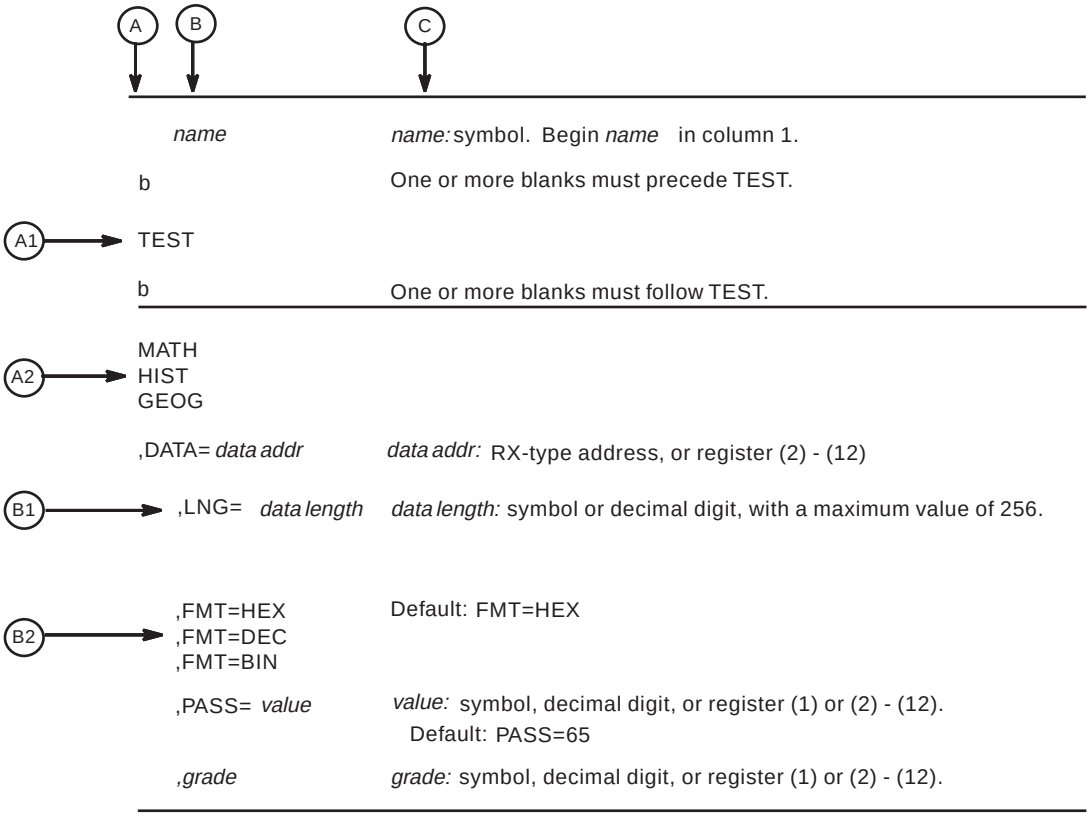

Figure 2. Sample Macro Syntax Diagram

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

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
ABEND	Yes	Yes	Yes	Yes
ALESERV	Yes	Yes	No	No
ASASYMBM	No	No	Yes	No
ATTACH	Yes (See note 1 on page 22)	No	Yes	No
ATTACHX	Yes	No	Yes	Yes
BLDMPB	Yes	Yes	No	No
BLSABDPL	Yes	Yes	N/A	No
BLSACBSP	Yes	Yes	N/A	No
BLSADSY	Yes	Yes	N/A	No
BLSAPCQE	Yes	Yes	N/A	No
BLSQFXL	Yes	Yes	N/A	No
BLSQMDEF	Yes	Yes	N/A	No
BLSQMFLD	Yes	Yes	N/A	No
BLSQSHDR	Yes	Yes	N/A	No
BLSRDRPX	Yes	Yes	N/A	No
BLSRESSY	Yes	Yes	N/A	No
BLSRNAMP	Yes	Yes	N/A	No
BLSRPRD	Yes	Yes	N/A	No
BLSRPWHS	Yes	Yes	N/A	No
BLSRSASY	Yes	Yes	N/A	No
BLSRXMSP	Yes	Yes	N/A	No
BLSRXSSP	Yes	Yes	N/A	No
BLSUPPR2	Yes	Yes	N/A	No
CALL	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
CHAP	No	No	No	Yes
CONVCON	No	No	No	No
CONVTOD	Yes	Yes	No	No
CPOOL	No	Yes	No	No
CPUTIMER	No	Yes	No	No
CSRCESRV	Yes	Yes	No	No
CSRCMPSC	Yes	Yes	Yes	No
CSREVW	No	No	N/A	No
CSRIDAC	No	No	N/A	No
CSRL16J	No	No	N/A	No
CSRPACT	Yes	Yes	N/A	No
CSRPBLD	Yes	Yes	N/A	No
CSRPCON	Yes	Yes	N/A	No
CSRPDAC	Yes	Yes	N/A	No
CSRPDIS	Yes	Yes	N/A	No
CSRPEXP	Yes	Yes	N/A	No
CSRPFRE	Yes	Yes	N/A	No
CSRPFR1	Yes	Yes	N/A	No
CSRPGET	Yes	Yes	N/A	No
CSRPGT1	Yes	Yes	N/A	No
CSRPQCL	Yes	Yes	N/A	No
CSRPQEX	Yes	Yes	N/A	No
CSRPQPL	Yes	Yes	N/A	No
CSRPRFR	Yes	Yes	N/A	No
CSRPRFR1	Yes	Yes	N/A	No
CSRPRGT	Yes	Yes	N/A	No
CSRPRGT1	Yes	Yes	N/A	No
CSRREFR	No	No	N/A	No
CSRSAVE	No	No	N/A	No
CSRSCOT	No	No	N/A	No
CSRSI	No	Yes	No	No
CSRUNIC	Yes	Yes	No	No
CSRVIEW	No	No	N/A	No
CSVAPF	Yes (See note 7 on page 22)	Yes	Yes	No
CSVINFO	No	No	No	No
CSVQUERY	Yes	No	Yes	Yes
DELETE	No	No	No	Yes
DEQ	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
DETACH	Yes	No	Yes	No
DIV	Yes	No	Yes	No
DOM	No	No	No	Yes
DSPSERV	Yes	Yes	Yes	Yes
EDTINFO	Yes	Yes	Yes	No
ENQ	No	No	No	Yes
ESPIE	No	No	No	Yes
ESTAE (See note 2 on page 22)	No	No	Yes	No
ESTAEX	Yes	Yes	Yes	Yes
EVENTS	No	No	No	No
FREEMAIN	No (See note 3 on page 22)	Yes	Yes	Yes
GETMAIN	No (See note 3 on page 22)	Yes	Yes	Yes
GQSCAN	No	Yes	No	No
HSPSERV	Yes	Yes (See note 4 on page 22)	(See note 5 on page 22)	No
IARR2V	Yes	Yes	No	No
IARVSERV	Yes	Yes	Yes	No
IARV64	Yes	Yes	Yes	Yes
IDENTIFY	No	No	No	Yes
IEAARR	Yes	Yes	Yes	No
IEABRC	Yes	Yes	N/A	No
IEAINTKN	Yes	Yes	Yes	No
IEALSQRY	Yes	Yes	Yes	No
IEANTCR	Yes	Yes	N/A	No
IEANTDL	Yes	Yes	N/A	No
IEANTRT	Yes	Yes	N/A	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
IEFDDSRV	Yes	Yes	No	No
IEFSSI	Yes	No	No	No
IOCINFO	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
IOSCHPD	Yes	Yes	Yes	No
ITZEVENT	No	Yes	No	No
ITZQUERY	No	Yes	No	No
IXGBRWSE	Yes	Yes	Yes	No
IXGCONN	Yes	Yes	Yes	No
IXGDELET	Yes	Yes	Yes	No
IXGINVNT	Yes	Yes	Yes	No
IXGWRITE	Yes	Yes	Yes	No
LINK	Yes (See note 1 on page 22)	No	Yes	No
LINKX	Yes	No	Yes	Yes
LOAD	Yes	No	No	Yes
LSEXPAND	Yes	No	No	No
PGLOAD	No	No	No	No
PGOUT	No	No	No	No
PGRLSE	No	No	No	No
PGSER	No	No	No	Yes
POST	No	Yes	No	Yes
QRYLANG	Yes	Yes	No	No
REFPAT	Yes	No	Yes	No
RESERVE	No	No	No	Yes
RETURN	No	No	No	No
SAVE	No	No	No	No
SETRP	Yes	Yes	Yes	Yes
SNAP	Yes (See note 1 on page 22)	No	Yes	No
SNAPX	Yes	No	Yes	No
SPIE	No	No	No	No
SPLEVEL	Yes	Yes	No	No
STAE	No	No	No	No
STATUS	Yes	Yes	No	No
STCKCONV	Yes	Yes	No	No
STCKSYNC	Yes	Yes	Yes	No
STIMER	No	No	No	Yes
STIMERM	No	No	No	Yes
STORAGE	Yes	Yes	No	Yes
SYMRBLD	Yes	Yes	Yes	No
SYMREC	No	Yes	No	No

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
SYNCH	Yes (See note 1 on page 22)	No	Yes	No
SYNCHX	Yes	No	Yes	Yes
SYSSTATE	Yes	Yes	No	No
TCBTOKEN	Yes	Yes	No	No
TESTART	Yes	Yes	No	No
TIME	Yes (See note 6 on page 22)	Yes (See note 6 on page 22)	No	Yes
TIMEUSED	Yes	Yes	No	Yes
TRANMSG	Yes	Yes	No	No
TTIMER	No	No	No	Yes
UCBDEVN	No	No	No	No
UCBINFO	Yes	Yes	Yes	No
UCBSCAN	Yes	Yes	Yes	No
UPDTMPB	Yes	Yes	No	No
VRADATA	Yes	Yes	Yes	No
WAIT	No	Yes	No	Yes
WTL	No	No	No	No
WTO	No	No	No	Yes
WTOR	No	No	No	Yes
XCTL	Yes (See note 1 on page 22)	Yes	Yes	Yes
XCTLX	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

Notes:

1. Callers can use either macro in the following macro pairs:
   ATTACH or ATTACHX
   LINK or LINKX
   SNAP or SNAPX
   SYNCH or SYNCHX
   XCTL or XCTLX

   IBM recommends that all callers in AR mode use the X-macros (ATTACHX, LINKX, SNAPX, SYNCHX, and XCTLX). If a program in AR mode issues ATTACH, LINK, SNAP, SYNCH, or XCTL after issuing SYSSTATE ASCENV=AR, the system substitutes the corresponding X-macro and issues a message telling you that it made the substitution.
2. The only programs that can use ESTAE are programs that are in primary mode with PASN=SASN=HASN. Callers in AR mode or in cross memory mode must use ESTAEX instead of ESTAE.

   IBM recommends you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, in which case, you should use ESTAE.
3. Problem state AR mode callers must use the STORAGE macro instead of using GETMAIN or FREEMAIN.
4. PASN=HASN=SASN for a non-shared standard hiperspace for which an ALET is not used (the HSPALET parameter is omitted).
5. If you use the HSPALET parameter, the HSPSERV macro checks SYSSTATE.
6. Only TIME LINKAGE=SYSTEM can be issued in AR mode, and can be issued in cross memory mode. TIME LINKAGE=SVC cannot be issued in AR mode or in cross memory mode.
7. For the QUERY request, CSVAPF can be issued only in primary mode. For all other requests, CSVAPF can be issued in primary or AR mode.

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: Assembler Services Guide*.

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-, 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
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.

© Copyright IBM Corp. 1988, 2002

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).
,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.

,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 4. 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.

Table 4. Return and Reason Codes for the IARR2V Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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
```

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: Assembler Services Guide* for more information about sharing data through the use of the IARVSERV macro.

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. See *z/OS MVS Programming: 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)*.
- The calling program can use IARVSERV only to share data that resides within the address space, or in a data space that the calling program created.

© Copyright IBM Corp. 1988, 2002

- 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.

Restrictions

None.

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 ,RETAIN=YES	**Default**: RETAIN=NO
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default**: IMPLIED_VERSION *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. The value specified must be no greater than 16 entries. 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 5. 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.

Table 5. Return and Reason Codes for the IARVSERV Macro (continued)

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

Table 5. Return and Reason Codes for the IARVSERV Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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.
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.
ƀ	One or more blanks must precede IARVSERV.
IARVSERV	
ƀ	One or more blanks must follow IARVSERV.

SHARE UNSHARE CHANGEACCESS	
,RANGLIST=*ranglist_addr*	*ranglist_addr*: RS-type address, or address in register (2) - (12).
,NUMRANGE=*numrange_addr*	*numrange_addr*: RS-type address, or address in register (2) - (12). **Default**: 1 range
,TARGET_VIEW=READONLY ,TARGET_VIEW=SHAREDWRITE ,TARGET_VIEW=UNIQUEWRITE ,TARGET_VIEW=TARGETWRITE ,TARGET_VIEW=LIKESOURCE ,TARGET_VIEW=HIDDEN	
,COPYNOW	
,RETAIN=NO	**Default**: RETAIN=NO

,RETAIN=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

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

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

Specifies the execute form of the IARVSERV macro.

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

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.

IARV64 — 64–Bit Virtual Storage Allocation

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 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 42)

PAGEOUT
: Notify the system that data within physical pages of one or more memory objects will not be used in the near future. (Page 48)

PAGEIN
: Notify the system that data within physical pages of one or more memory objects are needed in the near future. (Page 51)

DISCARDDATA
: Discard data within physical pages of one or more memory objects. (Page 55)

CHANGEGUARD
: Request that a specified range in a memory object be changed from the guard state to the usable state or vice versa. (Page 60)

DETACH
: Free one or more memory objects. (Page 64)

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

After the separate descriptions of each individual Request are the following sections which apply to all of the Requests:
- The abend codes on page 69,
- The return and reason codes on page 70, and
- Examples of using IARV64 on page 70

Note: The examples apply to REQUEST=GETSTOR and DETACH.

© Copyright IBM Corp. 1988, 2002

Facts associated with these services:

- A segment represents one megabyte of virtual storage 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 a 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 A 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:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

None.

Restrictions

No subspace or data space ALETs can be specified.

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.
ƀ	One or more blanks must precede IARV64.
IARV64	
ƀ	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).
,FPROT=**YES** ,FPROT=NO	**Default:** FPROT=YES
,SVCDUMPRGN=**YES** ,SVCDUMPRGN=NO	**Default:** SVCDUMPRGN=YES
,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** ,GUARDLOC=HIGH	**Default:** GUARDLOC=LOW
,TTOKEN=*ttoken*	*ttoken:* RS-type address or address in register (2) - (12).
,TTOKEN=**NO_TTOKEN**	**Default:** TTOKEN=NO_TTOKEN
,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** ,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=GETSTOR
A required parameter. REQUEST=GETSTOR creates a memory object. 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.

,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.

,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. Which is used is an application choice.

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 areas do 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 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.

,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=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:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

None.

Restrictions

No subspace or data space ALETs can be specified.

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).
,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 hence 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 is as follows:

VSA denotes the starting address of the data to be acted on.

The address specified must be within a created memory object returned by GETSTOR

The value provided 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.

,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:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

None.

Restrictions

No subspace or data space ALETs can be specified.

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).
,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=PAGEIN
A required parameter. REQUEST=PAGEIN notifies the system that data within the specified ranges is needed in the near future and should be, if possible, retrieved from auxillary storage. An attempt to PAGEIN a range which contains a guard area will cause an ABEND.

The caller must be in supervisor state OR system (0-7) PSW key OR be in a PSW key which matches the key of the memory object storage to be paged out.

,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 is as 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 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.

,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:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

None.

Restrictions

No subspace or data space ALETs can be specified.

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.
b	One or more blanks must precede IARV64.
IARV64	
b	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).
,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=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. However, 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 is as follows:

VSA denotes the starting address of the data to be acted on.

The address specified must be within a created memory object returned by GETSTOR

The value provided 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 the 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.

,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:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

None.

Restrictions

No subspace or data space ALETs can be specified.

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
,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 number of usable areas to the guard areas. The data in the converted pages will be released. This operation reduces the amount of virtual storage that contributes toward the MEMLIMIT for the address space.

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

When GUARDLOC=HIGH was 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.

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

When GUARDLOC=HIGH was 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.

,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 segments 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=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.
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:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

None.

Restrictions

No subspace or data space ALETs can be specified.

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**	**Default:** OWNER=YES
,TTOKEN=*ttoken*	*ttoken:* RS-type address or address in register (2) - (12).
,TTOKEN=**NO_TTOKEN**	**Default:** TTOKEN=NO_TTOKEN
,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** ,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=DETACH
A required parameter. REQUEST=DETACH frees one or more memory objects. Note that problem state programs running in PSW key (8-15) can use this function only when primary = home.

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 be 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. Note that memory objects not associated with a user token are not affected. (Such objects would have to have been created using GETSTOR NOUSERTKN). 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 to be freed.

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
An optional parameter that specifies whether the owning task still exists The default is OWNER=YES.

,OWNER=YES
The owning task must still exist (it may be in termination however). The TTOKEN is provided or defaulted for the owning task.

,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.

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.

,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 and Abend Reason 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 6. Return and Reason Codes for the IARV64 Macro

Hexadecimal Return Code	Hexadecimal 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 pertainining to those recent IARV64 requesets.
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 calls 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. 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
*
*************************************************************
* 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
         IARV64 MF=(L,V64L)
```

IDENTIFY — Add an Entry Name

Description

The IDENTIFY macro is used to add an entry name to a copy of a load module currently in virtual storage. The copy must be one of the following:

- A copy that satisfied the requirements of a LOAD macro issued during the execution of the current task.
- The last load module given control, if control was passed to the load module using a LINK, LINKX, ATTACH, ATTACHX, XCTL, or XCTLX macro.

Attention: The IDENTIFY macro may not be issued by an asynchronous exit routine. The IDENTIFY macro assigns the identified entry point as reentrant. Callers issuing this macro should be sure that their programs have been marked as reenterable.

The IDENTIFY service sets the addressing mode of the entry name that was added equal to the addressing mode of the major entry name. The system assigns the major entry name according to how the load module was constructed.

- If the load module was constructed using the linkage editor (and brought into virtual storage via program fetch or virtual fetch, the major entry name is the name of the load module in the partitioned data set directory (not an alias to that member).
- If the load module was brought into storage by the loader, the major entry name is either the name that the user provided as input to the loader or the name that the loader used as a default.

If an authorized caller creates an entry name for a module in the link pack area, the IDENTIFY service places an entry for the alias on the active link pack area queue. If an unauthorized caller creates an entry name for a module in the link pack area, the IDENTIFY service places an entry for the alias on the task's job pack queue.

If an unauthorized caller creates an entry name for an authorized module, the IDENTIFY service marks the new entry as unauthorized. In all other cases, the new entry name receives the same level of authorization as the main entry point.

The caller cannot have an EUT FRR established.

Syntax

The IDENTIFY macro is written as follows:

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

© Copyright IBM Corp. 1988, 2002

EP=*entry name*	*entry name*: Symbol
EPLOC=*entry name addr*	*entry name addr*: RX-type address, or register (0) or (2) - (12).
,ENTRY=*entry addr added*	*entry addr added*: RX-type address, or register (1) or (2) - (12).

Parameters

The parameters are explained as follows:

EP=*entry name*
EPLOC=*entry name addr*
Specifies the entry name or address of the entry name. The name does not have to correspond to any symbol or name in the load module, and must not correspond to any name, alias, or added entry name for a load module in the link pack area queue, or the job pack area of the job step. If EPLOC is coded, the name must be padded to eight bytes, if necessary.

,ENTRY=*entry addr added*
Specifies the virtual storage address of the entry point being added. If the program that issues the IDENTIFY macro is running in 24-bit addressing mode, the value for *entry addr added* must be a 24-bit address.

Return Codes

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Return Code	Meaning
00	Successful completion of requested function.
04	Entry name and address already exist.
08	Entry name duplicates the major name of a load module currently in virtual storage; entry address was not added.
0C	Entry address is not within an eligible load module; entry address was not added.
10	Request issued by an asynchronous exit routine; entry address was not added.
14	Entry name duplicates the name already used for a minor entry or for an entry created by another IDENTIFY request, and the entry point addresses are different; the current request is rejected.
18-1C	An internal error occurred. Record the return code and contact the appropriate IBM support personnel.
24	An unexpected error occurred.
28	The address specified by the EPLOC parameter was fetch protected.
2C	An internal error occurred. Record the return code and contact the appropriate IBM support personnel.
30	Unsuccessful processing due to a system queue area (SQA) storage shortage.
34	Unsuccessful processing due to a local system queue area (LSQA) storage shortage.

Hexadecimal Return Code	Meaning
38	Unsuccessful processing due an error in the job pack area. Record the return code and contact the appropriate IBM support personnel.

Example

Add an entry name (PGMTAL2A) to a load module in virtual storage. Register 3 contains the entry point address.

```
IDENTIFY     EP=PGMTAL2A,ENTRY=(R3)
```

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.
ƀ	One or more blanks must precede IEAARR.
IEAARR	
ƀ	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
   ...
```

IEABRC — Relative Branch Macro

Description

The IEABRC macro defines macros to intercept and change various base-displacement branch instructions to their relative branch equivalents. Many OS/390 macros contain base-displacement branches that could functionally be relative branches. In order to write an assembler routine that both uses these macros and uses relative branching, you can use IEABRC to enable those macros to use relative branches. Changing base-displacement branch instructions to their relative branch equivalents can eliminate code addressability issues.

Note: Using IEABRC does not guarantee that all OS/390 macros can be invoked without code addressability. Some macros still require addressability to the location where the macro is invoked.

Environment

Because IEABRC is not an executable macro, there are no specific environment requirements.

Programming Requirements

None.

Restrictions

IEABRC converts branch condition instructions to relative branch condition instructions except when both of the following conditions are true:

- The branch target ends with ″)″
- A ″(″ in the 2nd or subsequent characters of the branch target is not preceded by ″+″ or ″–″

For example:

`B X(15)`	Remains a base/displacement branch
`B X+(15)`	Converted to a relative branch
`B X+Y`	Converted to a relative branch

Register Information

Because IEABRC is not an executable macro, there is no need to save and restore register contents.

Performance Implications

None.

Syntax

The IEABRC macro is written prior to any base/displacement branch that needs to be converted to a relative branch as follows:

---

b One or more blanks must precede COPY.

© Copyright IBM Corp. 1988, 2002

COPY IEABRC

ƀ One or more blanks must follow IEABRC.

Parameters

IEABRC has no parameters of its own.

Example

The following example converts a base/displacement branch to a relative branch:

```
TEST     CSECT
R12      EQU   12
         USING STATICAREA,R12
         COPY  IEABRC
         ENQ   (QNAME,RNAME,E,RNAMELEN,SYSTEM)
STATICAREA DC  D'0'
QNAME    DC    CL8'THEQNAME'
RNAME    DC    CL8'THERNAME'
RNAMELEN EQU   L'RNAME
         END   TEST
```

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 7. 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

IEAINTKN — Build Incident Token

Description

Use the IEAINTKN macro to build an incident token. You can pass the token to other routines to identify related pieces of problem data.

Normally you will not need to use an IEAINTKN macro because the system generates an incident token when an SVC dump is requested and an incident token is not provided. For example, the system provides an incident token when it processes an SDUMPX macro without an INTOKEN parameter.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state with PSW key 8-15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- bit
ASC mode:	Primary or access register (AR)
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.

Programming Requirements

- Place the TOKEN area in the primary address space or, for AR-mode callers, in an address space or data space that is addressable through an ALET that you provide.
- Include the CVT mapping macro.

Restrictions

None.

Input Register Information

Before issuing the IEAINTKN 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
14-15	Used as work registers 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

© 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 standard form of the IEAINTKN macro is written as follows:

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

,TOKEN=*inctoken addr*	*inctoken addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

TOKEN=*inctoken addr*
Specifies the address of a 32-character area where the system builds the incident token. The area must begin on a doubleword boundary.

ABEND Codes

None.

Return and Reason Codes

None.

Example

Provide an incident token in the area named MYTOKEN.

```
         IEAINTKN TOKEN=MYTOKEN
         .
         .
         .
         DS  0D         Align parameter on double word boundary
MYTOKEN  DS  CL32       Incident token
         CVT ,          CVT mapping
```

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: Assembler Services Guide*.

The output of IEALSQRY depends upon the current environment and on the recovery environment that exists:

- If at least one ESTAE-type recovery routine is in effect, the output depends on the most recently activated routine:
  - If it is a STAE or STAI routine, a return code of 8 is returned.
  - 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 STAEs, ESTAEXs, ESTAEs exist for this RB and no ESTAI or STAIs are in effect, a return code of 8 is returned.

See *z/OS MVS Programming: 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
Cross memory mode:	Any PASN, any HASN, any SASN
Amode:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled
Interrupt status:	Enabled or disabled
Locks:	No locks are required.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

None.

© Copyright IBM Corp. 1988, 2002

Restrictions

None.

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.
ƀ	One or more blanks must precede IEALSQRY.
IEALSQRY	
ƀ	One or more blanks must follow IEALSQRY.

	Valid parameters
TOKEN=*token*	*token*: RS-type address or register (1) - (12). **Default**: Leave token in GPR 0.
,RETCODE=*retcode*	*retcode*: RS-type address, or register (2) - (12). **Default**: No retcode processing.

The parameters are explained as follows:

TOKEN=*token*
Specifies a halfword area (or the address of the area in register (1)-(12)) where the system places a value that indicates the difference between the number of linkage stack entries present when the recovery routine was activated and the number that are currently present. A recovery routine can place this value in field SDWALSLV (in mapping macro IHASDWA) to ensure that the retry routine runs with the proper level of the linkage stack. If you do not use TOKEN, you can find the value in GPR 0.

RETCODE=*retcode*
Specifies a fullword output variable (or register (2)-(12)) into which the system copies the return code GPR 15. If you do not use RETCODE, you can find the return code in GPR 15.

ABEND Codes

The IEALSQRY caller might receive abend code X'B78'. For detailed abend code information, see *z/OS MVS System Codes*.

Return Codes

When control returns to the caller, register 15 contains one of the following decimal return codes (hexadecimal values are shown in parentheses):

Table 8. 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.
8 (8)	**Meaning**: No recovery routine of the proper type exists. Either no recovery routine exists or the most recently activated recovery routine is STAE or STAI. **Action**: Avoid using the token when retrying. You cannot retry to the current linkage stack level.
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
```

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
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 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

None.

© 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.

,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 zero.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
00	0	**Meaning**: The operation was successful. **Action**: None.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
04	4	**Meaning**: The *user_name* specified already exists. **Action**: Choose a different *user_name*.
08	8	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 31-bit addressing mode.
10	16	**Meaning**: An unauthorized caller attempted to create a system-level name/token pair. **Action**: Check which level of name/token pair you are creating.
18	24	**Meaning**: The caller held locks. **Action**: Release all locks before issuing IEANTCR.
1C	28	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid values are 1, 2, or 3.
20	32	**Meaning**: The caller specified an incorrect *user_name*. **Action**: Respecify the correct *user_name*.
24	36	**Meaning**: The caller specified an incorrect *persist_option*. **Action**: You must specify zero for the *persist_option*.
28	40	**Meaning**: The caller was in AR ASC mode and AR1 was not zero. **Action**: Change your program to primary mode or make sure the parameter list is in the primary address space.
40	64	**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 PROGRAM'S
*                                     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 and any PSW key **Note:** Problem-state programs with PSW key 8 - 15 cannot delete name/token pairs created by supervisor-state or PSW key 0 - 7 programs.
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 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

None.

© Copyright IBM Corp. 1988, 2002

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.

,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 to 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 and decimal, tells what each means, and recommends an action that you should take:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
00	0	**Meaning**: The operation was successful. **Action**: None.
04	4	**Meaning**: The request is rejected because the system could not find the requested name/token pair. **Action**: Check the *user_name* you specified.
08	8	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 31-bit addressing mode.
10	16	**Meaning**: An unauthorized caller attempted to delete a system-level name/token pair or a name/token pair created by an authorized program. **Action**: Check which level of name/token pair you are deleting.
18	24	**Meaning**: The caller held locks. **Action**: Release all locks before issuing IEANTDL.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
1C	28	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid values are 1, 2, or 3.
20	32	**Meaning**: The caller specified an incorrect *user_name*. **Action**: Respecify the correct *user_name*.
28	40	**Meaning**: The caller was in AR ASC mode and AR1 was not zero. **Action**: Change your program to primary mode or make sure the parameter list is in the primary address space.
40	64	**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. For example, you can use IEANTRT to obtain the name of the logrec recording medium, which is 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 and any PSW key
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 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 106 for the list of definitions IFBNTASM provides.

Restrictions

Do not call the IEANTRT callable service with the *user_name* and *user_token* parameters in the same storage location.

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.

```
CALL IEANTRT  |
              |,(level
              |,user_name
              |,user_token
              |,return_code)
              |
```

Link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB unless you use one of the following techniques as an alternative to CALL IEANTRT:

```
1.   LOAD EP=IEANTRT
     Save the entry point address
     ...
     Put the saved entry point address into R15
     CALL (15),(...)

2.   L     15,X'10'
     L     15,X'220'(15,0)
     L     15,X'14'(15,0)
     L     15,X'08'(15,0)
     CALL  (15),(...)
```

This second technique requires AMODE=31, and, before the CALL is issued, verification that the 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 from which you want to retrieve the token:
- 1 - Task
- 2 - Home address space
- 3 - Primary address space
- 4 - System.

,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 and Reason 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 and decimal, tells what each means, and recommends an action that you should take:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
00	0	**Meaning**: The operation was successful. **Action**: None.
04	4	**Meaning**: The request is rejected because the system could not find the requested name/token pair. **Action**: Check the *user_name* you specified.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
08	8	**Meaning**: The request is rejected because the caller is in 24-bit addressing mode. **Action**: Change your program to 31-bit addressing mode.
1C	28	**Meaning**: The caller specified an incorrect *level*. **Action**: Respecify the correct *level*. Valid values are 1, 2, 3, or 4.
40	64	**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 IEANTCR 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
```

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.

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.

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.

© Copyright IBM Corp. 1988, 2002

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, 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).
,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 ,SDATA=GRSQ ,SDATA=LPA ,SDATA=LSQA ,SDATA=NUC ,SDATA=RGN ,SDATA=SQA ,SDATA=SUM ,SDATA=SWA ,SDATA=TRT ,SDATA=PSA	**Default**: SDATA=DEFS
,ASYNC=**NO** ,ASYNC=YES	**Default**: ASYNC=NO
,ECBAD=*ecbad*	*ecbad*: RS-type address or register (2) - (12).

,ECB=*ecb*	*ecb*: RS-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,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**) ,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 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.

,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 9. 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.
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.

Table 9. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 9. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 9. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 9. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 9. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 9. Return and Reason Codes for the IEATDUMP Macro (continued)

Return Code	Reason Code	Meaning and Action
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: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: PASN=SASN=HASN
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.

Input Register Information

Before calling Allocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	Address of the parameter address list.
13	Address of a 72-byte register save area.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system

© Copyright IBM Corp. 1988, 2002

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 authorization level of the pause element being allocated. This is the minimum auth_level that can be specified when services such as Pause and Release use this pause element. 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	Authorization level of the PE being allocated.

,pause_element_token

Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element which you can use to synchronize the processing of a task.

ABEND Codes

None.

Return Codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: **Decimal (Hex)** **Equate symbol**	**Meaning and Action**
00 (0) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. 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_PE_NOT_HOME	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_XFER_TO_SELF	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
48 (30) IEA_XFER_FAILED	**Meaning**: Environmental error. The system could not obtain storage for a pause element. The system rejects the service call. **Action**: Retry the request later. If the problem persists, consult your system programmer.
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: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: PASN=SASN=HASN
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 auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input Register Information

Before calling Deallocate_Pause_Element, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	Address of the parameter address list.
13	Address of a 72-byte register save area.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

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

© Copyright IBM Corp. 1988, 2002

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.

```
CALL IEAVDPE  (return_code
              ,auth_level
              ,pause_element_token)
```

Parameters

The parameters are explained as follows:

return_code
Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Deallocate_Pause_Element service.

,auth_level
Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

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

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	This pause element being deallocated must have been allocated with auth_level=IEA_UNAUTHORIZED.

,pause_element_token
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element that is no longer needed.

ABEND Codes

None.

Return Codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (00) IEA_SUCCESS	**Meaning**: Successful completion **Action**: None.
04 (04)	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
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 specified 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.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

IEAVPSE — Pause Service

Description

Call Pause to make the current task nondispatchable. Once you pause a task, it remains nondispatchable until a Release service specifying the same PET is called. That is, the program issuing the Pause does not receive control back until after the Release occurs.

If a Release service specifying the same PET is called before Pause, the system returns control immediately to the calling program, and the task is not paused.

When you use Pause, it returns an updated PET; you use this updated PET to either deallocate or reuse the PE.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: PASN=SASN=HASN
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 is running auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only pause another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input Register Information

Before calling the Pause service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	Address of the parameter address list.
13	Address of a 72-byte register save area.

© 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 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 level that the specified pause element was allocated with. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The pause element being paused must have been allocated with auth_level=IEA_UNAUTHORIZED.

,pause_element_token
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A pause element token that identifies the pause element being used to pause the current task. You obtain the PET from the Allocate_Pause_Element service.

Once you use a PET in a call to the Pause service, you cannot reuse the PET on a second call to Pause or on a call to Transfer. The Pause service returns a new PET in updated_pause_element_token. The new PET now identifies the pause element used to Pause the task; use the new PET the next time you make a Pause request using the same Pause element.

,updated_pause_element_token
Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

A new pause element token that identifies the pause element originally identified by the PET specified in pause_element_token, which cannot be reused after a successful call to Pause.

,release_code
Returned parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

The release code, specified by the issuer of the Release service. A Release that specified this code released the task from its paused condition.

ABEND Codes

Abend Code	Reason Code	Description
AC7	001A0001	This is an internal error. Contact IBM support.

Return Codes

When the service returns control to your program, GPR 15 contains one of the following return codes:

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
00 (00) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	**Meaning**: Program error. The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET 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.
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 the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

IEAVRLS — Release

Description

Call Release to remove a task that has been paused, or to keep a task from being paused. Although a pause element can be used multiple times to pause a task, a pause element token can be used to successfully pause and release a task only once. Each time a pause element is used, the system generates a new PET to identify the pause element. The system returns the new updated PET on calls to the Pause and Transfer services.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: PASN=SASN=HASN
AMODE: 31-bit addressing mode.
ASC mode: Primary mode.
Interrupt status: Enabled or disabled 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

When the calling program specifies auth_level=IEA_UNAUTHORIZED, the caller must be in task mode and can only release another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input Register Information

Before calling the Release service, the caller must ensure that the following general purpose (GPRs) contain the specified information:

Register	Contents
1	Address of the parameter address list.
13	Address of a 72-byte register save area.

Output Register Information

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

Register	Contents
0-1	Used as work registers by the system

© Copyright IBM Corp. 1988, 2002

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 that the specified pause element was allocated with. IEAASM and IEAC define constants IEA_UNAUTHORIZED and IEA_AUTHORIZED, which the calling program can use. The following levels are supported:

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The pause element being released must have been allocated with auth_level=IEA_UNAUTHORIZED.

,target_du_pause_element_token

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains the pause element token that identifies the pause element used to pause the task. If the PET identifies a pause element that has not been paused (that is, the task has not been paused), the task will not be paused. However, the value specified in target_du_release_code will be returned to the caller of Pause.

,target_du_release_code

Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the caller of Pause or Transfer service that used (or will use) the same PET to pause a task. If your program is not using this code for communication, set this field to zero.

ABEND Codes

None.

Return Codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return code in: **Decimal (Hex)** **Equate symbol**	**Meaning and Action**
00 (00) IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None.
04 (04) IEA_PE_TOKEN_BAD	**Meaning**: The specified pause element token is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
08 (08) IEA_PE_TOKEN_STALE	**Meaning**: The specified pause element token is stale; that is, it was valid but has been used on the Pause or Transfer service. This service requires the updated PET returned on Pause or Transfer. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
16 (10) IEA_SLEEP_DISRUPTED	**Meaning**: RTM has terminated the task; no release is necessary. **Action**: None
20 (14) IEA_SPACE_TERMINATING	**Meaning**: The address space that contains the task that is terminating; no release is necessary. **Action**: None

Return code in: Decimal (Hex) Equate symbol	Meaning and Action
24 (18) IEA_LOCK_HELD	**Meaning**: Program error. The caller is holding one o r more locks; no locks must be held. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
32 (20) IEA_PE_BAD_STATE	**Meaning**: Program error. The pause element associated with the pause element token specified is invalid or has already been prereleased. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
36 (24) IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28) IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C) IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C) IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40) IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
4095 (FFF) IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

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, and when possible, give it immediate control. This service can also, optionally, pause the task under which the Transfer request is made. If the caller does not request that its task be paused, the caller's task remains dispatchable.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: PASN=SASN=HASN
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 transfer to another task in its home address space. All pause element tokens (PETs) used when auth_level=IEA_UNAUTHORIZED must have been obtained using an authorization level of IEA_UNAUTHORIZED.

Input Register Information

Before calling the Transfer service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	Address of the parameter address list.
13	Address of a 72-byte register save area.

Output Register Information

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

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged

© Copyright IBM Corp. 1988, 2002

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

```
CALL IEAVXFR  (return_code
              ,auth_level
              ,current_du_pause_element_token
              ,updated_pause_element_token
              ,current_du_release_code
              ,target_du_pause_element_token
              ,target_du_release_code)
```

Parameters

The parameters are explained as follows:

return_code

Returned parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

Contains the return code from the Transfer service.

,auth_level

Supplied parameter
- Type: Integer
- Character Set: N/A
- Length: 4 bytes

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

Variable	Value (hexadecimal)	Meaning
IEA_UNAUTHORIZED	0	The pause elements must have been allocated with auth_level=_UNAUTHORIZED.

,current_du_pause_element_token

Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element used to pause the current task. Once a PET is used on a call to the Pause service, it cannot be reused on a second call to Pause or as a current_du_pause_element_token on Transfer. A new PET is returned to updated_pause_element_token. The new PET now properly defines the pause element and should be used the next time a pause, transfer, release, or deallocate_pause_element request is made using the same pause element.

If the value specified is 16-bytes of binary zeros, the current task will not be paused. The updated_pause_element_token and current_du_release_code will be unpredictable.

CAUTION:
Do not specify the same PET for both current_du_pause_element_token and target_pause_element_token.

,updated_pause_element_token

Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a new pause element token that identifies the pause element originally identified by the PET specified in current_du_pause_element_token. The PET originally specified in current_du_pause_element_token cannot be reused after a successful call to Pause or Transfer.

If you set the current_du_pause_element_token to zeros, the contents of updated_pause_element_token are unpredictable.

,current_du_release_code

Returned parameter

- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code set by the issuer of the Release or Transfer service that released the current task from its paused condition.

If you set the current_du_pause_element_token to zero, the contents are unpredictable.

,target_du_pause_element_token

Supplied parameter

- Type: Character string
- Character Set: N/A
- Length: 16 bytes

Contains a pause element token that identifies the pause element to release the target task. Any PET that specifies a pause element not currently being used to

pause a task is valid. When a PET for a previously released pause element is used to try to pause a task, the task is not paused; however, the value specified in target_du_release_code will still be returned to the caller of Pause or Transfer.

If the task was paused and is now dispatchable, the task will immediately be given control on the current processor.

CAUTION:
Do not use the same PET for both current_du_pause_element_token and target_du_pause_element_token.

,target_du_release_code
Supplied parameter
- Type: Character string
- Character Set: N/A
- Length: 3 bytes

Contains the release code returned to the issuer of the Pause or Transfer service that is used (or will use) the same PET to pause a task.

ABEND Codes

None.

Return Codes

When the service returns control to the resource manager, GPR 15 and return_code contain a hexadecimal return code.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
00 (00)	IEA_SUCCESS	**Meaning**: Successful completion. **Action**: None
04 (04)	IEA_LOCK_HELD	**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.
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.

Return Code in: Decimal (Hex)	Equate symbol	Meaning and Action
36 (24)	IEA_UNSUPPORTED_MVS_RELEASE	**Meaning**: Environmental error. The system release does not support this service. The system rejects the service call. **Action**: Run the program on a system that supports the service.
40 (28)	IEA_INVALID_AUTHCODE	**Meaning**: Program error. The auth_level value specified in the call is not valid. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
44 (2C)	IEA_INVALID_MODE	**Meaning**: Program error. The calling program is not in primary ASC mode, which this service requires. The system rejects the service call. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
60 (3C)	IEA_AUTH_TOKEN	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was allocated with auth_level=AUTHORIZED. The system rejects the service call. **Action**: Program error. The specified pause element token is not valid. The system rejects the service call.
64 (40)	IEA_PE_NOT_HOME	**Meaning**: Program error. The caller specified auth_level=UNAUTHORIZED, but the pause element token was for a pause element allocated to another address. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
68 (44)	IEA_XFER_TO_SELF	**Meaning:** Program error. The specified current_du_pause_element_token and target_du_pause_element_token are the same. **Action**: Check the calling program for a probable coding error. Correct the program and rerun it.
72 (48)	IEA_XFER_FAILED	**Meaning**: The transfer failed, and the current_du_pause_element_token is no longer useable. **Action**: Reissue the transfer request using the updated_du_pause_element_token. Deallocate the current_du_pause_element_token.
4095 (FFF)	IEA_UNEXPECTED_ERROR	**Meaning**: This service routine encountered an unexpected error. The system rejects this service request. **Action**: Contact IBM support.

IEFDDSRV — Receive Device Information For an Allocation Request

Description

Use the IEFDDSRV macro to obtain the unit control block (UCB) addresses of the devices that were allocated for an allocation request. When you specify that an above 16 megabyte UCB not be captured during dynamic allocation, use the IEFDDSRV macro to retrieve the UCB address. When you invoke IEFDDSRV, you can identify the DD request by specifying the ddname, data control block (DCB) pointer, data set association block (DSAB) pointer, or access method control block (ACB) pointer.

For guidance about obtaining UCB information, see *z/OS MVS Programming: Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31-bit
ASC mode: Primary or Access register (AR)
Interrupt status: Enabled for I/O and external interrupts.
Locks: No locks may be held.
Control parameters: Control parameters must be in the primary address space.

Programming Requirements

An authorized caller must provide or inherit serialization on the SYSZTIOT resource before calling the IEFDDSRV macro and while using its output addresses. For unauthorized callers, the IEFDDSRV service will obtain and release the necessary SYSZTIOT serialization on behalf of the caller.

The caller must include the IEFDISMP and IEFDISRC mapping macros.

Restrictions

The returned UCB addresses are only valid while the devices remain allocated after the invocation of IEFDDSRV.

Input Register Information

Before issuing the IEFDDSRV 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

© Copyright IBM Corp. 1988, 2002

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 IEFDDSRV macro is written as follows:

---

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

RETRIEVE	
,DEVENTRY	
	Note: Exactly one of the following four (a mutually exclusive set of keys), must be specified.
,DDNAME=*ddname*	*ddname*: RS-type address or register (2) - (12).
,DSABPTR=*dsabptr*	*dsabptr*: RS-type address or register (2) - (12).
,DCBPTR=*dcbptr*	*dcbptr*: RS-type address or register (2) - (12).
,ACBPTR=*acbptr*	*acbptr*: RS-type address or register (2) - (12).
,SUBPOOL=*subpool*	*subpool*: RS-type address or register (2) - (12). **Default**: SUBPOOL=0
,DEVAREA=*devarea*	*devarea*: RS-type address or register (2) - (12).
,TCBPTR=*tcbptr*	*tcbptr*: RS-type address or register (2) - (12).
,RETCODE=*retcode addr*	*retcode addr*: RS-type address or register (2) - (12) of fullword output variable.

,RSNCODE=*rsncode addr*	*rsncode addr*: RS-type address or register (2) - (12) of fullword output variable.
MF=(S)	**Default**: S

Parameters

The parameters are explained as follows:

RETRIEVE
Specifies that you want to retrieve DD related information.

,DEVENTRY
Specifies that you want to obtain the UCB address for the devices allocated to the request.

,DDNAME=*ddname*
One of a set of mutually exclusive keys, it specifies the ddname of the DD statement.

,DSABPTR=*dsabptr*
One of a set of mutually exclusive keys, it specifies the address of the DSAB associated with a DD name.

,DCBPTR=*dcbptr*
One of a set of mutually exclusive keys, it specifies the address of the DCB associated with a DD name. When the data set associated with the DCB is open, the DD information is provided for the current task. Therefore, anything you specify on the TCBPTR parameter is ignored. When the data set is closed, you could be requesting the DD information for any task; therefore, use the TCBPTR parameter to indicate the task control block (TCB).

,ACBPTR=*acbptr*
One of a set of mutually exclusive keys, it specifies the address of the ACB associated with a DD name. When the data set associated with the ACB is open, the DD information is provided for the current task. Therefore, anything you specify on the TCBPTR parameter is ignored. When the data set is closed, you could be requesting the DD information for any task; therefore, use the TCBPTR parameter to indicate the TCB.

,SUBPOOL=*subpool*
Specifies the subpool for the device output area.

If your program's PSW key is different than the TCB key, specify a subpool that enables the IEFDDSRV macro to obtain storage in the same key as your program. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about how to select the right subpool.

,DEVAREA=*devarea*
Specifies the address of the area of storage to contain the output from the macro. The output is an array of device entry lists. The array contains information about the devices allocated for an allocation request. Each device entry list contains the number of devices and the UCB addresses of those devices. The IEFDISMP mapping macro maps the output. IBM recommends

that you set the *devarea* to zero before invoking IEFDDSRV; on return from the service, you can easily determine whether the service has an address in this field.

If IEFDDSRV returns with return code 0 and reason code 0, the system has obtained a storage area of the appropriate size in the requested key and subpool, and placed its address in *devarea*. You are responsible for releasing this storage. If the return code and reason codes are not 0, the system has not obtained the storage area; do not attempt to release the storage.

If you specify the DD name of a concatenation of data sets, the output area contains a device entry list for each data set in the concatenation. This situation is also true if you specified a DCB pointer or ACB pointer for a closed data set in a concatenation of data sets. If you specify the DSAB pointer, the DCB pointer, or ACB pointer for an open data set, the device output area contains information about just the specific single data set, even if the data set is in a concatenation.

,TCBPTR=tcbptr
Specifies the address of the TCB associated with the task for which the devices were allocated. When the data set associated with the DCB is open, IEFDDSRV provides the DD information for the current task.

,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

None.

Return and Reason Codes

When the IEFDDSRV 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. Return and reason codes are mapped in macro IEFDISRC. The hexadecimal return and reason codes from the IEFDDSRV macro are as follows:

Table 10. Return and Reason Codes for the IEFDDSRV Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: Successful completion. **Action**: None.
08	04	**Meaning**: Program error: incorrect input parameter. The specified or obtained DD name is blank. **Action**: Check the value you specified on the DDNAME parameter.
08	08	**Meaning**: Program error: incorrect input parameter. The specified or obtained DSAB pointer is zero. **Action**: Check the value you specified on the DSABPTR parameter.

Table 10. Return and Reason Codes for the IEFDDSRV Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	0C	**Meaning**: Program error: incorrect input parameter. The specified DCB pointer is zero. **Action**: Check the value you specified on the DCBPTR parameter.
08	10	**Meaning**: Program error: incorrect input parameter. An incorrect subpool was specified. **Action**: Check the value you specified on the SUBPOOL parameter.
08	14	**Meaning**: Program error: incorrect input parameter. The specified ACB pointer is zero. **Action**: Check the value you specified on the ACBPTR parameter.
0C	04	**Meaning**: Program error: The specified or obtained DD name is incorrect. **Action**: Check the value you specified on the DDNAME parameter.
0C	08	**Meaning**: Program error: The specified or obtained DSAB pointer is incorrect. **Action**: Check the value you specified on the DSABPTR parameter.
0C	0C	**Meaning**: System error: This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain a resource that is required. **Action**: Record the return code and supply it to the appropriate IBM support personnel.
0C	10	**Meaning**: System error: This return code is for IBM diagnostic purposes only. Most likely, the system could not obtain a lock that is required. **Action**: Record the return code and supply it to the appropriate IBM support personnel.
0C	14	**Meaning**: Program error: The specified TCB pointer is incorrect. **Action**: Check the value you specified on the TCBPTR parameter.
10	None	**Meaning**: System error: Recovery entered. **Action**: Check the dump produced by the abend and supply it to the appropriate IBM support personnel.

IEFDDSRV—List Form

Use the list form of the IEFDDSRV 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 IEFDDSRV macro is written as follows:

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

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 IEFDDSRV 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 IEFDDSRV 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.

IEFDDSRV—Execute Form

Use the execute form of the IEFDDSRV 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 IEFDDSRV macro is written as follows:

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

RETRIEVE

,DEVENTRY

	Note: Exactly one of the following four (a mutually exclusive set of keys), must be specified.
,DDNAME=*ddname*	*ddname*: RS-type or address in register (2) - (12).
,DSABPTR=*dsabptr*	*dsabptr*: RS-type or address in register (2) - (12).
,DCBPTR=*dcbptr*	*dcbptr*: RS-type or address in register (2) - (12).
,ACBPTR=*acbptr*	*acbptr*: RS-type or address in register (2) - (12).
,SUBPOOL=*subpool*	*subpool*: RS-type or address in register (2) - (12). **Default**: SUBPOOL=0
,DEVAREA=*devarea*	*devarea*: RS-type or address in register (2) - (12).
,TCBPTR=*tcbptr*	*tcbptr*: RS-type or address in register (2) - (12).
,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

The parameters are explained under the standard form of the IEFDDSRV 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 IEFDDSRV 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.

IEFPRMLB — Logical Parmlib Support

Description

The Logical Parmlib Concatenation is a set of up to 10 partitioned data sets defined by PARMLIB statements in the LOADxx member of either SYSn.IPLPARM or SYS1.PARMLIB which contains many initialization parameters in a pre-specified form in a single logical data set, thus minimizing the need for the operator to enter parameters. SYS1.PARMLIB makes the 11th or last data set in the concatenation and is the default logical parmlib if no PARMLIB statements exist in LOADxx.

The objective of this support is to allow installations to partition access to parmlib and isolate members customized by an installation from IBM maintenance and product level upgrades. The logical parmlib is established during IPL and is used by Master Scheduler Initialization and IEFPRMLB. There is a new SETLOAD command that allows you to switch from one logical parmlib to another without an IPL. The IEFPRMLB macro allows you to access the logical parmlib.

Use the IEFPRMLB macro to:
- Allocate the logical parmlib data set concatenation
- Unallocate the logical parmlib data set concatenation
- Read a logical parmlib data set
- Retrieve information about which data sets make up the logical parmlib

The four functions for the macro are:
- IEFPRMLB REQUEST=ALLOCATE allocates the logical parmlib via DDname.
- IEFPRMLB REQUEST=FREE unallocates the logical parmlib via DDname.
- IEFPRMLB REQUEST=LIST retrieves information about the logical parmlib data set concatenation.
- IEFPRMLB REQUEST=READMEMBER reads a specified member of an already allocated logical parmlib and returns its contents in an input buffer.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts will result.
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space.

Programming Requirements

The caller should include the IEFZPRC mapping macro to get return and reason code equates for all the functions.

If you are going to use the read, message or list buffers, then you should include the IEFZPMAP mapping macro to get their mappings.

© Copyright IBM Corp. 1988, 2002

Restrictions

The caller may not have an EUT FRR established.

Input Register Information

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

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code when GPR15 is not 0
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

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

Performance Implications

None.

REQUEST=ALLOCATE Option of IEFPRMLB

Syntax

The IEFPRMLB macro is written as follows:

---

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

REQUEST=ALLOCATE

,S99RB=NO ,S99RB=YES	**Default**: S99RB=NO
,WAITDSN=NO ,WAITDSN=YES	**Default**: WAITDSN=NO
,MOUNT=YES ,MOUNT=NO	**Default**: MOUNT=YES
,RETMSG=NO ,RETMSG=YES	**Default**: RETMSG=NO
,CONSOLID=*consolid* ,CONSOLID=**NOCONSID**	*consolid*: RS-type address or register (2) - (12). **Default**: CONSOLID=NOCONSID
,CART=*cart* ,CART=**NOCART**	*cart*: RS-type address or register (2) - (12). **Default**: CART=NOCART
,MSGBUF=*msgbuf* ,MSGBUF=**NOMSGBUF**	*msgbuf*: RS-type address or register (2) - (12). **Default**: MSGBUF=NOMSGBUF
,S99RBPTR=*s99rbptr*	*s99rbptr*: RS-type address or register (2) - (12).
,ALLOCDDNAME=*allocddname*	*allocddname*: RS-type address or register (2) - (12).
,READ=NO ,READ=YES	**Default**: READ=NO
,MEMNAME=*memname*	*memname*: RS-type address or register (2) - (12).
,READBUF=*readbuf*	*readbuf*: RS-type address or register (2) - (12).
,BLANK72=YES ,BLANK72=NO	**Default**: BLANK72=YES
,MEMNOTFOUND=MSGOK ,MEMNOTFOUND=NOMSG	**Default**: MEMNOTFOUND=MSGOK
,FREECLOSE=NO ,FREECLOSE=YES	**Default**: FREECLOSE=NO
,CALLERNAME=*callername*	*callername*: 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** ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default**: PLISTVER=IMPLIED_VERSION
,MF=**S** ,MF=(L,*list addr*)	**Default**: MF=S *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:

REQUEST=ALLOCATE
A required parameter. REQUEST=ALLOCATE allocates the logical parmlib data set concatenation. The allocation uses the data set name(s) and volume serial number(s) provided on the PARMLIB statements in the LOADxx member of SYSn.IPLPARM or SYS1.PARMLIB that is used during IPL processing or as specified by a SETLOAD command. If a volume serial number(s) isn't specified, IEFPRMLB searches the catalog for it. The allocation uses DISP=SHR and UNIT=SYSALLDA. If no PARMLIB statements are provided in the LOADxx member, the allocation uses only SYS1.PARMLIB.

,S99RB=NO
,S99RB=YES
An optional parameter, that specifies whether or not an SVC99 request block is input. The default is S99RB=NO.

,S99RB=NO
specifies that no S99RB is input.

,S99RB=YES
specifies that an SVC99RB (and optionally an SVC99RBX) is input. The SVC99 request block is only required when the caller requires S99FLAG1/S99FLAG2 options not automatically provided by the ALLOCATE function. If the caller requires that the allocation wait for data sets to become available or allow mounting of volumes, the caller must set the appropriate bits in the S99FLAG1/S99FLAG2 fields to request those options. The address of the list of text unit pointers (S99TXTPP) must be zero. If an SVC99 request block is passed and the caller wishes messages issued or returned, the caller must also provide an SVC99 request block extension. The SVC99 request block and SVC99 request block extension are mapped by mapping macro IEFZB4D0.

,WAITDSN=NO
,WAITDSN=YES
An optional parameter when S99RB=YES is not specified, that indicates whether waiting should be allowed for one or more of the data sets in the logical parmlib data set concatenation if they are not readily available (for example, enqueued exclusive by another job). The default is WAITDSN=NO.

,WAITDSN=NO
If one or more of the data sets in the logical parmlib data set concatenation is not readily available (e.g., enqueued exclusive by another job), waiting should not be allowed. In this case upon return from the IEFPRMLB service the logical parmlib data set concatenation will not have been allocated.

,WAITDSN=YES
If one or more of the data sets in the logical parmlib data set concatenation is not readily available (for example, enqueued exclusive by another job),

waiting should be allowed. In this case the service will wait for the data set(s) to become available before proceeding with the allocation. Upon return from the IEFPRMLB service the logical parmlib data set concatenation will have been allocated barring other errors.

,MOUNT=YES
,MOUNT=NO
An optional parameter when S99RB=YES is not specified, that indicates whether the service should allow mounting of volumes or consideration of offline or pending offline devices for one or more of the data sets in the logical parmlib data set concatenation. The default is MOUNT=YES.

,MOUNT=YES
If one or more of the volumes on which one or more of the data sets in the logical parmlib reside is not currently mounted, mounting of that volume(s) should be allowed. If one or more of the devices on which one or more of the data sets in the logical parmlib reside is not currently online or is pending offline, consideration of the offline or pending offline device should be allowed. Upon return from the IEFPRMLB service the logical parmlib data set concatenation will have been allocated barring other errors.

,MOUNT=NO
If one or more of the volumes on which one or more of the data sets in the logical parmlib reside is not currently mounted, mounting of that volume(s) should not be allowed. If one or more of the devices on which one or more of the data sets in the logical parmlib reside is not currently online, consideration of the offline device should not be allowed. Upon return from the IEFPRMLB service the logical parmlib data set concatenation will not have been allocated.

,RETMSG=NO
,RETMSG=YES
An optional parameter when S99RB=YES is not specified, that specifies whether or not messages are to be returned to the caller in an input message buffer. The default is RETMSG=NO.

,RETMSG=NO
specifies that messages generated during IEFPRMLB processing should not be returned to the caller in the input message buffer (MSGBUF). Messages generated during IEFPRMLB processing will be issued to the console specified by the input console id or will be issued with Route Code 11 (Programmer Information) and descriptor code 4 (System Status) if no console id is input.

,RETMSG=YES
specifies that messages generated during IEFPRMLB processing should be returned to the caller in the input message buffer (MSGBUF). Note that the only messages capable of being returned are those issued by MVS Allocation and SMS. Also, only error messages (severity level 8 and higher) are returned with RETMSG=YES. If warning messages (severity level 4) or informational messages (severity level 0) are desired, then an S99RB and an S99RBX with the desired message severity level (S99EMGSV) must be built and passed by specifying, S99RB=YES, MSGBUF=msgbuf, and S99RBPTR=s99rbptr.

,CONSOLID=*consolid*
,CONSOLID=NOCONSID
An optional input parameter when RETMSG=YES and S99RB=YES are not

specified. It contains the id of the console that originated this request and may be provided if messages are to be issued. The default is NOCONSID.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,CART=*cart*
,CART=NOCART
An optional input parameter when RETMSG=YES and S99RB=YES are not specified, that contains the command and response token. The default is NOCART.

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

,MSGBUF=*msgbuf*
,MSGBUF=NOMSGBUF
A required input parameter when RETMSG=YES is specified and S99RB=YES is not specified, that is the area into which all messages generated during IEFPRMLB processing are to be placed. The format of each message returned in the buffer is mapped by IEFZPMAP and is compatible with WTO format requirements for the TEXT keyword. There may be more than one message in the buffer. A 4K buffer is recommended. Messages are placed contiguously into the buffer in 256-byte message elements. If the input buffer is not large enough to contain all the generated messages, those messages that will fit are returned in the buffer in the order they are generated. If the message buffer is filled, an indicator (PRM_Msg_Buffer_Full) will be returned to indicate the buffer is full and, therefore, may not contain all messages. PRM_Message_Count will contain the number of messages in the buffer. See DSECT PRM_Message_Buffer in IEFZPMAP for a complete mapping of the message buffer.

The caller must fill in the following fields in the message buffer (DSECT PRM_Message_Buffer):
- PRM_Msg_Buffer_Size set to the size of the buffer (including the header)
- All other fields set to zero

The default is NOMSGBUF.

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

,S99RBPTR=*s99rbptr*
A required input parameter when S99RB=YES is specified that contains the address of the SVC99 request block to be used to process the allocation request.

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

,ALLOCDDNAME=*allocddname*
A required input output parameter, that is the DDname associated with the logical parmlib. If a non-blank/non-zero DDname is input, the service will examine the active task's TIOT to determine if the DDname is currently allocated. If it is currently allocated, the service will return to its caller without further processing. The service will set return code x'04' (PRMLB_WARNING) and reason code x'01' (PRMLB_DD_ALREADY_ALLOC) to indicate the DDname is currently allocated. If the DDname is not currently allocated, the service will allocate the logical parmlib data set concatenation using the input DDname.

If a blank or zero DDname is input, the service will allocate the logical parmlib data set concatenation and return the system-generated DDname to the caller.

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

,READ=NO
,READ=YES
An optional parameter, that specifies whether or not a specified member is to be read from the logical parmlib. The default is READ=NO.

,READ=NO
indicates that no read is to be performed.

,READ=YES
indicates that the specified member of the logical parmlib data set concatenation is to be read and placed into the input buffer. If READ is requested, the member to be read (specified by MEMNAME) and the buffer in which to place the member contents (specified by READBUF) must be provided.

,MEMNAME=*memname*
A required input parameter when READ=YES is specified, that is the name of the member which is to be read from the logical parmlib data set concatenation. The entire contents of the specified member will be read from the logical parmlib data set concatenation and returned in the input buffer specified on the READBUF keyword.

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

,READBUF=*readbuf*
A required input output parameter when READ=YES is specified, that is the area into which the contents of the member of the logical parmlib data set concatenation (specified by MEMNAME) are to be placed. The format of the buffer is mapped by IEFZPMAP. If the member is too large to fit into the buffer, records will be read into the buffer until the buffer is full. The service will terminate with return code x'0C' (PRMLB_Request_Failed) and reason code x'0A' (PRMLB_Read_Buffer_Full) and upon return, the buffer header will contain the buffer size needed to contain the entire member contents. The caller may obtain a larger buffer and invoke IEFPRMLB to read the member again from the beginning. The read buffer header will also contain the number of records that were successfully read the placed into the input buffer and the total number of records contained in the specified member.

For each record read, columns 73 - 80 will be blanked. Unless requested by the Blank72 parameter, column 72 will also be blanked. Symbolic substitution will be performed.

The caller must fill in the following fields in the READ buffer (DSECT PRM_Read_Buffer):
- PRM_Read_BuffSize - set to the size of the buffer
- All other fields set to zero

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

,BLANK72=YES
,BLANK72=NO
An optional parameter when READ=YES is specified, that indicates whether or

not to blank out column 72. Most parmlib processing is defined to ignore column 72. The default is BLANK72=YES.

,BLANK72=YES
Do blank out column 72.

,BLANK72=NO
Do not blank out column 72.

,MEMNOTFOUND=MSGOK
,MEMNOTFOUND=NOMSG
An optional keyword input thta indicates whether or not to write a message when the member is not found. The default is MEMNOTFOUND=MSGOK.

,MEMNOTFOUND=MSGOK
Specifies to write a message.

,MEMNOTFOUND=NOMSG
Specifies not to write a message..

,FREECLOSE=NO
,FREECLOSE=YES
An optional keyword input that indicates whether the “logical parmlib” dataset concatenation should be automatically unallocated when the DD is closed. The default is FREECLOSE=NO.

,FREECLOSE=NO
The “logical parmlib” dataset concatenation will not be automatically unallocated when the DD is closed. When the caller’s use of the “logical parmlib” dataset concatenation has been complete, the caller must reinvoke the IEFPRMLB service with REQUEST=FREE to unallocated the “logical parmlib” dataset concatenation. Additionally, the caller must ensure the “logical parmlib” has been closed prior to reinvoking the IEFPRMLB service with REQUEST=FREE.

,FREECLOSE=YES
The “logical parmlib” dataset concatenation will be automatically unallocated when the DD is closed. The caller does not need to reinvoke the IEFPRMLB service with REQUEST=FREE. However, the caller should be aware that the “logical parmlib” dataset concatentaion will be automatically unallocated as soon as it is closed and would therefore no longer be allocated for use by the caller.

Note: If the caller requests READ(YES) and FREECLOSE(YES), the caller does not need to close the data set nor reinvoke the IEFPRMLB service to free the “logical parmlib” dataset concatenation. The close and free will be done by the Logical Parmlib Service.

,CALLERNAME=*callername*
A required input parameter, that is the EBCDIC caller’s name which is to be used in messages, symptom records and other diagnostic areas as necessary during IEFPRMLB processing. Initial characters A-I and SYS are reserved for IBM use.

The suggested callername definition is ’ProgramName || ServiceLevel’

Example:

```
IEF761I jjobname [procstep] stepname ddname callername
        DD IS ALREADY ALLOCATED AND WILL BE USED BY
        THIS TASK
```

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

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

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=*plistver*
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **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 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, 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, 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.

ABEND Codes

None.

Return and Reason Codes

When the IEFPRMLB 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.

Return and reason code constants are defined in macro IEFZPRC.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

Table 11. Return and Reason Codes for the IEFPRMLB Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
X’00’	—	**Equate Symbol**: PRMLB_Success **Meaning**: Return Code - function completed successfully **Action**: None required.
X’04’	—	**Equate Symbol**: PRMLB_Warning **Meaning**: Return Code - Warning
X’04’	X’01’	**Equate Symbol**: PRMLB_DD_Already_ALLOC **Meaning**: The specified DDname is already allocated to this task. **Action**: None required.

Table 11. Return and Reason Codes for the IEFPRMLB Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
X'08'	—	**Equate Symbol**: PRMLB_Locks_Held **Meaning**: Return Code - the caller of IEFPRMLB holds a lock. **Action**: Change the caller's code to release locks prior to invoking IEFPRMLB.
X'0C'	—	**Equate Symbol**: PRMLB_Request_Failed **Meaning**: Return Code - request failed.
X'0C'	X'01'	**Equate Symbol**: PRMLB_Member_Not_Found **Meaning**: The specified member name was not found. **Action**: Ensure the specified member name exists. If so, contact the system programmer.
X'0C'	X'02'	**Equate Symbol**: PRMLB_Read_IO_Error **Meaning**: An I/O error was encountered while attempting to read the specified member. **Action**: Contact the system programmer.
X'0C'	X'03'	**Equate Symbol**: PRMLB_Open_Error **Meaning**: An error was encountered while attempting to open the logical parmlib. **Action**: Contact the system programmer.
X'0C'	X'04'	**Equate Symbol**: PRMLB_ALLOC_Failed **Meaning**: Allocation of one of the logical parmlib data sets failed **Action**: Contact the system programmer.
X'0C'	X'05'	**Equate Symbol**: PRMLB_CONCAT_Failed **Meaning**: Concatenation of the logical parmlib data sets failed **Action**: Contact the system programmer.
X'0C'	X'06'	**Equate Symbol**: PRMLB_Reader_Load_Failed **Meaning**: Load of the parmlib read routine failed. **Action**: Contact the system programmer.
X'0C'	X'07'	**Equate Symbol**: PRMLB_Unable_To_Access_DS **Meaning**: The parmlib read routine was unable to access the logical parmlib **Action**: Contact the system programmer.

Table 11. Return and Reason Codes for the IEFPRMLB Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
X'0C'	X'08'	**Equate Symbol**: PRMLB_Parmlib_Still_Open **Meaning**: REQUEST=FREE was requested but the logical parmlib is still open. **Action**: Close the data set prior to issuing the REQUEST=FREE.
X'0C'	X'09'	**Equate Symbol**: PRMLB_UNALLOC_Failed **Meaning**: Unallocation of the logical parmlib data sets failed. **Action**: Contact the system programmer.
X'0C'	X'0A'	**Equate Symbol**: PRMLB_Read_Buffer_Full **Meaning**: The input READ buffer is full and READ processing could not continue **Action**: The caller may obtain a buffer large enough to contain the entire member contents (PRM_Buff_Size_Needed in DSECT PRM_Read_Buffer which is mapped by IEFZPMAP contains the required size) and re-invoke IEFPRMLB to begin reading the specified member again.
X'0C'	X'0B'	**Equate Symbol**: PRMLB_Putline_Error **Meaning**: Putline processing abended. This could be due to an error in the user-provided CPPL (pointed to by S99ECPPL when the user provides an S99RB). **Action**: Verify that the CPPL is valid.
X'10'	—	**Equate Symbol**: PRMLB_Internal_Error **Meaning**: Return Code - an internal error occurred.
X'10'	X'01'	**Equate Symbol**: PRMLB_Bad_Parameter **Meaning**: A bad parameter list was passed to the parmlib read routine. **Action**: Contact the system programmer.
X'10'	X'02'	**Equate Symbol**: PRMLB_Unknown_Reason **Meaning**: Return Code - Reason for failure is unknown. **Action**: Contact the system programmer.
X'14'	—	**Equate Symbol**: PRMLB_Not_Task_Mode **Meaning**: Return Code - the caller is not in Task mode. **Action**: Contact the system programmer.
X'1C'	—	**Equate Symbol**: PRMLB_Invalid_Parameter_List **Meaning**: Return Code - the input parameter list is invalid.

Table 11. Return and Reason Codes for the IEFPRMLB Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
X'1C'	X'01'	**Equate Symbol**: PRMLB_Plist_Unaccessible **Meaning**: The IEFPRMLB service was unable to access the input parameter list. **Action**: Ensure the parameter list resides in storage belonging to the caller. If so, contact the system programmer.
X'1C'	X'02'	**Equate Symbol**: PRMLB_ListBuff_Unaccessible **Meaning**: The IEFPRMLB service was unable to access the input LIST buffer. **Action**: Ensure the list buffer resides in storage belonging to the caller. If so, contact the system programmer.
X'1C'	X'03'	**Equate Symbol**: PRMLB_MsgBuff_Unaccessible **Meaning**: The IEFPRMLB service was unable to access the input message buffer. **Action**: Ensure the message buffer resides in storage belonging to the caller. If so, contact the system programmer.
X'1C'	X'04'	**Equate Symbol**: PRMLB_ReadBuff_Unaccessible **Meaning**: The IEFPRMLB service was unable to access the input read buffer. **Action**: Ensure the read buffer resides in storage belonging to the caller. If so, contact the system programmer.
X'1C'	X'05'	**Equate Symbol**: PRMLB_Plist_S99TXTPP_NOT0 **Meaning**: The S99RB provided to the IEFPRMLB service contains a non-zero S99TXTPP field. **Action**: Change the caller's code to zero the S99TXTPP prior to the call to IEFPRMLB.
X'1C'	X'06'	**Equate Symbol**: PRMLB_MsgBuff_Format_Error **Meaning**: The format of the message buffer provided to the IEFPRMLB service is invalid. **Action**: Correct the message buffer format.
X'1C'	X'07'	**Equate Symbol**: PRMLB_ReadBuff_Format_Error **Meaning**: The format of the read buffer provided to the IEFPRMLB service is invalid. **Action**: Correct the read buffer format.
X'1C'	X'08'	**Equate Symbol**: PRMLB_ListBuff_Format_Error **Meaning**: The format of the list buffer provided to the IEFPRMLB service is invalid. **Action**: Correct the list buffer format.

Table 11. Return and Reason Codes for the IEFPRMLB Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
X'1C'	X'09'	**Equate Symbol**: PRMLB_S99RB_Unaccessible **Meaning**: The IEFPRMLB service was unable to access the input read buffer. **Action**: Ensure the S99RB resides in storage belonging to the caller. If so, contact the system programmer.
X'20'	—	**Equate Symbol**: PRMLB_Cross_Memory **Meaning**: Return Code - the caller is in cross memory mode. **Action**: Change the caller's code so it is not in cross memory mode when invoking IEFPRMLB.
X'24'	—	**Equate Symbol**: PRMLB_ESTAE_Setup_Failed **Meaning**: Return Code - a failure occurred when IEFPRMLB processing attempted to set up an ESTAE environment. **Action**: Contact the system programmer.
X'28'	—	**Equate Symbol**: PRMLB_Notauth_To_Subpool **Meaning**: Return Code - an unauthorized caller requested messages in an authorized subpool. **Action**: Only specify subpools to which the program is authorized.

REQUEST=FREE Option of IEFPRMLB

Syntax

The IEFPRMLB macro is written as follows:

---

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

REQUEST=FREE

,RETMSG=NO ,RETMSG=YES	**Default**: RETMSG=NO

,CONSOLID=*consolid*	*consolid*: RS-type address or register (2) - (12).
,CONSOLID=**NOCONSID**	**Default**: CONSOLID=NOCONSID
,CART=*cart*	*cart*: RS-type address or register (2) - (12).
,CART=**NOCART**	**Default**: CART=NOCART
,MSGBUF=*msgbuf*	*msgbuf*: RS-type address or register (2) - (12).
,MSGBUF=**NOMSGBUF**	**Default**: MSGBUF=NOMSGBUF
,DDNAME=*ddname*	*ddname*: RS-type address or register (2) - (12).
,CALLERNAME=*callername*	*callername*: 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=*plistver*	
,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:

REQUEST=FREE
A required parameter. REQUEST=FREE unallocates the logical parmlib data set concatenation.

,RETMSG=NO
,RETMSG=YES
An optional parameter, that indicates whether or not messages are to be returned to the caller in an input message buffer. The default is RETMSG=NO.

,RETMSG=NO
specifies that messages generated during IEFPRMLB processing should not be returned to the caller in the input message buffer (MSGBUF). Messages generated during IEFPRMLB processing will be issued to the console specified by the input console id or will be issued with Route Code 11 (Programmer Information) and descriptor code 4 (System Status) if no console id is input.

,RETMSG=YES
specifies that messages generated during IEFPRMLB processing should be returned to the caller in the input message buffer (MSGBUF). Note that the

only messages capable of being returned are those issued by MVS Allocation and SMS. Also, only error messages (severity level 8 and higher) are returned with RETMSG=YES.

,CONSOLID=*consolid*
,CONSOLID=NOCONSID
An optional input parameter when RETMSG=YES is not specified, that contains the id of the console which originated this request and may be provided if messages are to be issued. The default is NOCONSID.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,CART=*cart*
,CART=NOCART
An optional input parameter when RETMSG=YES is not specified, that contains the Command And Response Token. The default is NOCART.

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

,MSGBUF=*msgbuf*
,MSGBUF=NOMSGBUF
A required input parameter when RETMSG=YES is specified, that is the area into which all messages generated during IEFPRMLB processing are to be placed. The format of each message returned in the buffer is mapped by IEFZPMAP and is compatible with WTO format requirements for the TEXT keyword. There may be more than one message in the buffer. A 4K buffer is recommended. Messages are placed contiguously into the buffer in 256-byte message elements. If the input buffer is not large enough to contain all the generated messages, those messages that will fit are returned in the buffer in the order they are generated. If the message buffer is filled, an indicator (PRM_Msg_Buffer_Full) will be returned to indicate the buffer is full and, therefore, may not contain all messages. PRM_Message_Count will contain the number of messages in the buffer. See DSECT PRM_Message_Buffer in IEFZPMAP for a complete mapping of the message buffer.

The caller must fill in the following fields in the message buffer (DSECT PRM_Message_Buffer):
- PRM_Msg_Buffer_Size set to the size of the buffer (including the header)
- All other fields set to zero

The default is NOMSGBUF.

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

,DDNAME=*ddname*
A required input parameter, that is the DDname associated with the logical parmlib. The logical parmlib data set concatenation will be unallocated. The DDname originally input to or returned by the invocation of IEFPRMLB REQUEST=ALLOCATE should be input. If the logical parmlib is open when IEFPRMLB is invoked with REQUEST=FREE, the unallocation will fail.

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

,CALLERNAME=*callername*
A required input parameter, that is the EBCDIC caller's name which is to be

used in messages, symptom records and other diagnostic areas as necessary during IEFPRMLB processing. Initial characters A-I and SYS are reserved for IBM use.

The suggested callername definition is 'ProgramName || ServiceLevel'

Example:

```
IEF761I jjobname [procstep] stepname ddname callername
        DD IS ALREADY ALLOCATED AND WILL BE USED BY
        THIS TASK
```

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

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

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=*plistver*
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **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 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, 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, 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.

ABEND Codes

None.

Return and Reason Codes

When the IEFPRMLB 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.

See the return codes in under REQUEST=ALLOCATE option of IEFPRMLB.

Examples

None.

REQUEST=LIST Option of IEFPRMLB

Syntax

The IEFPRMLB macro is written as follows:

---

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

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

REQUEST=LIST	
,BUFFER=*buffer*	*buffer:* RS-type address or register (2) - (12).
,CALLERNAME=*callername*	*callername:* 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** ,PLISTVER=MAX ,PLISTVER=*plistver*	**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:

REQUEST=LIST

A required parameter. REQUEST=LIST requests information about the logical parmlib data set concatenation. For each data set included in the logical parmlib, for which there is room in the provided buffer, the following information is returned:

- Data set name (either specified on a PARMLIB statement in LOADxx or SYS1.PARMLIB (if no PARMLIB statements are provided in LOADxx)).
- Volume serial number where the data set resides (if a volume serial number is provided on the PARMLIB statement).

The number of data sets which make up the logical parmlib data set concatenation is also returned. If this number is larger than the number of 60-byte entries for which room was provided, then the system did not return all of the available information. In that case, you should allocate a larger buffer based on the returned number and call the service again, in order to retrieve all of the information.

NOTE: The LIST function only returns information on those data sets which are currently being used by the system. If a data set was found unusable during LOADxx processing, that data set is not being used as part of the logical parmlib concatenation and its information will not be returned by the LIST function. Exclusion of unusable data sets is only possible when no SETLOAD command was issued after IPL since an unusable data set encountered during SETLOAD processing causes SETLOAD to fail.

,BUFFER=*buffer*
A required input parameter, that is the area where the information about the logical parmlib data set concatenation is to be placed. The buffer is mapped by IEFZPMAP. The caller must fill in the following fields in the list buffer (DSECT PRM_List_Buffer):

- PRM_List_Version
  - Set this using either equate symbol PRM_List_VER1 or PRM_List_Current_Version.
- PRM_List_Buff_Size
  - Set this to the size of the provided area. It must be at least the size of PRM_List_Header. It should contain room for at least 11 60-byte entries as well.
- All other fields set to zero

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

,CALLERNAME=*callername*
A required input parameter, that is the EBCDIC caller's name which is to be used in messages, symptom records and other diagnostic areas as necessary during IEFPRMLB processing. Initial characters A-I and SYS are reserved for IBM use.

The suggested callername definition is 'ProgramName || ServiceLevel'

Example:

```
IEF761I jjobname [procstep] stepname ddname callername
        DD IS ALREADY ALLOCATED AND WILL BE USED BY
        THIS TASK
```

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

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

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=*plistver*
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When

using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **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 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, 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, 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.

ABEND Codes

None.

Return and Reason Codes

When the IEFPRMLB 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.

See return codes under REQUEST=ALLOCATE option of IEFPRMLB.

Examples

None.

REQUEST=READMEMBER Option of IEFPRMLB

Syntax

The IEFPRMLB macro is written as follows:

---

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

REQUEST=READMEMBER	
,DDNAME=*ddname*	*ddname*: RS-type address or register (2) - (12).
,MEMNAME=*memname*	*memname*: RS-type address or register (2) - (12).
,READBUF=*readbuf*	*readbuf*: RS-type address or register (2) - (12).
,BLANK72=YES ,BLANK72=NO	**Default**: BLANK72=YES
,MSG=YES ,MSG=NO	**Default**: MSG=YES
,RETMSG=NO ,RETMSG=YES	**Default**: RETMSG=NO

,CONSOLID=*consolid*	*consolid*: RS-type address or register (2) - (12).
,CONSOLID=**NOCONSID**	**Default**: CONSOLID=NOCONSID
,CART=*cart*	*cart*: RS-type address or register (2) - (12).
,CART=**NOCART**	**Default**: CART=NOCART
,MSGBUF=*msgbuf*	*msgbuf*: RS-type address or register (2) - (12).
,MSGBUF=**NOMSGBUF**	**Default**: MSGBUF=NOMSGBUF
,CALLERNAME=*callername*	*callername*: 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=*plistver*	
,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:

REQUEST=READMEMBER
A required parameter. REQUEST=READMEMBER indicates to read the specified member of the logical parmlib data set concatenation and place the contents into the input buffer.

,DDNAME=*ddname*
A required input parameter, that is the DDname associated with the allocated logical parmlib.

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

,MEMNAME=*memname*
A required input parameter, that is the name of the member which is to be read from the logical parmlib data set concatenation. The entire contents of the specified member will be read from the logical parmlib data set concatenation and returned in the input buffer specified on the READBUF keyword.

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

,READBUF=*readbuf*
A required input output parameter, that is the area into which the contents of the member of the logical parmlib data set concatenation (specified by MEMNAME) are to be placed. The format of the buffer is mapped by IEFZPMAP. If the

member is too large to fit into the buffer, records will be read into the buffer until the buffer is full. The service will terminate with return code x'0C' (PRMLB_Request_Failed), reason code x'0A' (PRMLB_Read_Buffer_Full) and upon return, the buffer header will contain the buffer size needed to contain the entire member contents. The caller may obtain a larger buffer and invoke IEFPRMLB to read the member again from the beginning. The read buffer header will also contain the number of records that were successfully read the placed into the input buffer and the total number of records contained in the specified member.

For each record read, columns 73 - 80 will be blanked. Unless requested by the Blank72 parameter, column 72 will also be blanked. Symbolic substitution will be performed.

The caller must fill in the following fields in the READ buffer (DSECT PRM_Read_Buffer):
- PRM_Read_BuffSize - set to the size of the buffer
- All other fields set to zero

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

,BLANK72=YES
,BLANK72=NO
An optional parameter, that indicates whether or not to blank out column 72. Most parmlib processing is defined to ignore column 72. The default is BLANK72=YES.

,BLANK72=YES
Do blank out column 72.

,BLANK72=NO
Do not blank out column 72.

,MSG=YES
,MSG=NO
An optional parameter, that indicates whether or not message processing is to be performed. The default is MSG=YES.

,MSG=YES
specifies that message processing is to be performed.

,MSG=NO
specifies that no message processing is to be performed. If MSG=NO is coded, no messages generated by the logical parmlib service will be issued to the console or hardcopy log and no messages will be returned to the caller.

,RETMSG=NO
,RETMSG=YES
An optional parameter when MSG=YES is specified, that indicates whether or not messages are to be returned to the caller in an input message buffer. The default is RETMSG=NO.

,RETMSG=NO
specifies that messages generated during IEFPRMLB processing should not be returned to the caller in the input message buffer (MSGBUF). Messages generated during IEFPRMLB processing will be issued to the console specified by the input console id or will be issued with Route Code 11 (Programmer Information) and descriptor code 4 (System Status) if no console id is input.

,RETMSG=YES
specifies that messages generated during IEFPRMLB processing should be returned to the caller in the input message buffer (MSGBUF). Note that the only messages capable of being returned are those issued by MVS Allocation and SMS. Also, only error messages (severity level 8 and higher) are returned with RETMSG=YES.

,CONSOLID=*consolid*
,CONSOLID=NOCONSID
An optional input parameter when RETMSG=YES is not specified and MSG=YES is specified, that contains the id of the console which originated this request and may be provided if messages are to be issued. The default is NOCONSID.

To code: Specify the RS-type address, or address in register (2)-(12), of a 4-character field.

,CART=*cart*
,CART=NOCART
An optional input parameter when RETMSG=YES is not specified and MSG=YES is specified, that contains the Command And Response Token. The default is NOCART.

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

,MSGBUF=*msgbuf*
,MSGBUF=NOMSGBUF
A required input parameter when RETMSG=YES and MSG=YES are specified, that is the area into which all messages generated during IEFPRMLB processing are to be placed. The format of each message returned in the buffer is mapped by IEFZPMAP and is compatible with WTO format requirements for the TEXT keyword. There may be more than one message in the buffer. A 4K buffer is recommended. Messages are placed contiguously into the buffer in 256-byte message elements. If the input buffer is not large enough to contain all the generated messages, those messages that will fit are returned in the buffer in the order they are generated. If the message buffer is filled, an indicator (PRM_Msg_Buffer_Full) will be returned to indicate the buffer is full and, therefore, may not contain all messages. PRM_Message_Count will contain the number of messages in the buffer. See DSECT PRM_Message_Buffer in IEFZPMAP for a complete mapping of the message buffer.

The caller must fill in the following fields in the message buffer (DSECT PRM_Message_Buffer):
- PRM_Msg_Buffer_Size set to the size of the buffer (including the header)
- All other fields set to zero

The default is NOMSGBUF.

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

,CALLERNAME=*callername*
A required input parameter, that is the EBCDIC caller's name which is to be used in messages, symptom records and other diagnostic areas as necessary during IEFPRMLB processing. Initial characters A-I and SYS are reserved for IBM use.

The suggested callername definition is 'ProgramName || ServiceLevel'

Example:

```
IEF761I jjobname [procstep] stepname ddname callername
        DD IS ALREADY ALLOCATED AND WILL BE USED BY
        THIS TASK
```

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

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15.

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=*plistver*
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **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 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, 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, 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.

ABEND Codes

None.

Return and Reason Codes

When the IEFPRMLB 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.

See return codes under REQUEST=ALLOCATE option of IEFPRMLB.

Examples

None.

IEFSSI — Dynamically Query a Subsystem

Description

Use the IEFSSI macro to dynamically query a subsystem. The REQUEST=QUERY parameter allows an application to query the following information for all subsystems defined to the SSI:

- The subsystem name
- If the subsystem is dynamic or not dynamic
- If the subsystem is the primary subsystem
- If the subsystem is active or inactive
- If the subsystem is dynamic, whether it accepts or rejects the SETSSI command
- If the subsystem is active, which function codes it supports.
- The number of vector tables associated with the subsystem, with a maximum of two vector tables.
- The following information for each associated vector table:
  - If the vector table is managed by the SSI. A vector table managed by the SSI is a vector table created with the IEFSSVT REQUEST=CREATE macro.
  - A locator. This locator is a token if the vector table is is managed by the SSI and is an address if the vector table is not managed by the SSI.
  - If the vector table is active
  - The function codes supported by the vector table

This information represents a snapshot of the subsystems defined to the SSI when you process the query request.

To obtain information about the primary subsystem without knowing its name, use the query request and specify a subsystem name of '!PRI'.

Environment

The requirements for the caller are:

Minimum authorization:	For the QUERY request, problem state with any PSW key.
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.
- Include the IEFJSQRY macro to map the REQUEST=QUERY output.

Restrictions

The caller must not have established an EUT FRR.

© Copyright IBM Corp. 1988, 2002

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=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.
ƀ	One or more blanks must precede IEFSSI.
IEFSSI	
ƀ	One or more blanks must follow IEFSSI.

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

,WORKAREA=*workarea*	*workarea*: RS-type address or register (2) - (12) of an output area.
,WORKASP=*workasp*	*workasp*: RS-type address or 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 register (2) - (12). of fullword output variable
,RSNCODE=*rsncode*	*rsncode*: RS-type address or 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 12. 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: • QUERY — Information for all subsystems defined to the SSI has been queried **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.
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)	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.

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

Return Code decimal (hex)	Reason Code decimal (hex)	Meaning and Action
12 (0C)	900 (384)	**Equate Symbol**: IEFSSI_QUERY_STORAGE **Meaning**: Unable to obtain storage for an output of the QUERY request. **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. See *z/OS MVS Programming: Authorized Assembler Services Reference ENF-IXG* for more information on the IEFSSI macro.
20 (14)	—	**Equate Symbol**: IEFSSI_SYSTEM_ERROR **Meaning**: System error **Action**: Investigate the following possible causes: • Inability to obtain a system resource • Abnormal task termination Obtain the system dump, if any, and contact the IBM support center.
24 (18)	—	**Equate Symbol**: IEFSSI_UNAVAILABLE **Meaning**: The IEFSSI macro has been invoked too early during system initialization. **Action**: Delay the invocation of the IEFSSI macro to a later point in the IPL.

Example

Obtain subsystem information for any subsystem whose name begins with 'JES' and free the storage returned by the system.

```
         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
```

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.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- bit
ASC mode:	Primary or access register (AR)
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any
Control parameters:	Must be in the primary address space or be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming Requirements

If in AR mode, specify SYSSTATE ASCENV=AR before invoking the macro.

Restrictions

None.

Input Register Information

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

Output Register Information

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

Register	Contents
0	Reason code if GPR 15 contains a return code of 08; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

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

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

© Copyright IBM Corp. 1988, 2002

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).
,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.

,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).
,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.

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 13. 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 13. 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.

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.

ITTUINIT — Activate external CTRACE recording

Description

> **Note**
> ITTUINIT is a linkable system service.

ITTUINIT is one of a set of services that an unauthorized program can use to write CTRACE output. The other services in the set are ITTUWRIT and ITTUTERM. The services must be invoked under the same task in problem state.

Use the ITTUINIT service to activate external CTRACE recording. Once ITTUINIT has been invoked, multiple calls to the ITTUWRIT service can be made to write the CTRACE entries. The ITTUTERM service is invoked to end external CTRACE recording.

The caller of ITTUINIT provides a data structure containing parameters for the service. At the conclusion of its processing, ITTUINIT returns information for the user in the same data structure.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state with PSW key 8-15
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

1. To build the parameter area required by ITTUINIT, you must include the ITTUIPRM mapping macro (see *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*).
2. Before calling ITTUINIT, the caller must provide the following in the fully-initialized ITTUIPRM mapping macro:
   - The component name
   - The name of the format table for the component
   - The ddname to which CTRACE output is to be written
   - The maximum length of an ITTCTE that will be accepted by ITTUWRIT.
   - The number of bytes of virtual storage that the unauthorized CTRACE writer is authorized to use for trace buffers.
   - An option bit that requests NOWRAP processing. WRAP processing is requested when this bit is zero.

   DSECT=NO may be specified for initial values.
3. The caller determines whether ITTUINIT processing was successful by examining the return code.

© Copyright IBM Corp. 1988, 2002

4. Successful ITTUINIT processing results in the following updated field in ITTUIPRM:
   - A token whose value must be passed to the ITTUWRIT and ITTUTERM services.

Restrictions

The caller cannot have any enabled, unlocked task (EUT) FRRs established.

Input Register Information

Before linking to ITTUINIT, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	Address of the parameter list
13	Address of a standard 72-byte save area in the primary address space

Before linking to ITTUINIT, the caller does not have to place any information into any access register (AR).

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as 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

Use the following form of the LINK macro to invoke the ITTUINIT service:

```
►►──label──┬─LINK──EP=ITTUINIT──,MF=(E,parmarea)──────────────────┬──────────────────────────────►◄
           └─LINKX──EP=ITTUINIT──,MF=(E,parmarea)──┬──────────────┬┘
                                                   └─,SF=(E,parmlist)─┘
```

Note: As an alternative to using LINK or LINKX, callers in 31-bit AMODE can also:

1. Issue the MVS LOAD macro to load the ITTUINIT service and obtain its entry point address.

2. Issue the CALL macro to call the service. Specify MF=(E,*your_parmlist*) on the call.

Parameters

The parameters are explained as follows:

label
The name on the macro invocation.

LINK
LINKX
Names the system service that is to be used for linkage.

EP=ITTUINIT
Specifies the entry point name for the ITTUINIT service.

,MF=(E,*parmarea*)
Specifies the address of the parameter list to be passed to ITTUINIT. The parameter list consists of the following address:

- The address of the fully-initialized ITTUIPRM.

,SF=(E,*parmlist*)
For use with LINKX when your program is reentrant. Before you call LINKX with this parameter, define *parmlist* using the LIST form of LINKX.

Return and Reason Codes

When the ITTUINIT service returns control to your program, Register 15 contains a return code.

Decimal Return Code	Meaning and Action
00	**Meaning**: The ITTUINIT request completed successfully. **Action**: None required.
16	**Meaning**: Warning. The ITTUINIT request did not complete successfully. **Action**: Reissue ITTUINIT.

ITTUTERM — End external CTRACE recording

Description

> **Note**
> ITTUTERM is a linkable system service.

Use the ITTUTERM service to close the trace data set and unallocate resources that were allocated by the ITTUINIT service. uffer.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state with PSW key 8-15
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

1. The caller determines whether ITTUTERM processing was successful by examining the return code.

Restrictions

The caller cannot have any enabled, unlocked task (EUT) FRRs established.

Input Register Information

Before linking to ITTUTERM, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register	Contents
1	Address of parameter list
13	Address of a standard 72-byte save area in the primary address space

Before linking to ITTUTERM, the caller does not have to place any information into any access register (AR).

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as 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

Use the following form of the LINK macro to invoke the ITTUTERM service:

```
►►─label──┬─LINK──EP=ITTUTERM──,MF=(E,parmarea)──────────────────┬──────────────────────────►◄
          └─LINKX──EP=ITTUTERM──,MF=(E,parmarea)──┬──────────────┬┘
                                                  └─,SF=(E,parmlist)┘
```

Note: As an alternative to using LINK or LINKX, callers in 31-bit AMODE can also:

1. Issue the MVS LOAD macro to load the ITTUTERM service and obtain its entry point address.
2. Issue the CALL macro to call the service. Specify MF=(E,*your_parmlist*) on the call.

Parameters

The parameters are explained as follows:

label
The name on the macro invocation.

LINK
LINKX
Names the system service that is to be used for linkage.

EP=ITTUTERM
Specifies the entry point name for the ITTUTERM service.

,MF=(E,*parmarea*)
Specifies the address of the parameter list to be passed to ITTUTERM. The parameter list consists of the following address:

- The address of the token passed from ITTUINIT.

,SF=(E,*parmlist*)
For use with LINKX when your program is reentrant. Before you call LINKX with this parameter, define *parmlist* using the LIST form of LINKX.

Return and Reason Codes

When the ITTUTERM service returns control to your program, Register 15 contains a return code.

Decimal Return Code	Meaning and Action
00	**Meaning**: The ITTUTERM request completed successfully. **Action**: None required.
non-zero	**Meaning**: Warning. The ITTUTERM request did not complete successfully. **Action**: Reissue ITTUTERM.

ITTUWRIT — Queue a group of CTRACE entries

Description

> **Note**
> ITTUWRIT is a linkable system service.

ITTUWRIT is one of a set of services that an unauthorized program can use to write CTRACE output. The other services in the set are ITTUINIT and ITTUTERM. The services must be invoked under the same task in problem state.

Use the ITTUWRIT service to queue a group of CTRACE entries. Whenever new CTRACE entries overflow a buffer, recording of the entries occurs.

The caller of ITTUWRIT provides the token returned by the ITTUINIT service and the address of the storage area containing the ITTCTE entries.

Multiple calls to the ITTUWRIT service can be made to write the CTRACE entries. When ITTUWRIT is in control, the system writes the ITTCTE entries from the storage area passed to ITTUWRIT into the CTRACE output buffers immediately. If necessary, the system may need to discard trace entries because of timing considerations or error conditions such as I/O errors or storage overlays. ITTUWRIT adds control information to the trace data set whenever data losses occur, if possible.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state with PSW key 8-15
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

1. To reference the parameter area required by ITTUWRIT, you must include the ITTUIPRM mapping macro (see *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*).
2. The caller determines whether ITTUWRIT processing was successful by examining the return code.

Restrictions

The caller cannot have any enabled, unlocked task (EUT) FRRs established.

Input Register Information

Before linking to ITTUWRIT, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

© Copyright IBM Corp. 1988, 2002

Register	Contents
1	Address of parameter list
13	Address of a standard 72-byte save area in the primary address space

Before linking to ITTUWRIT, the caller does not have to place any information into any access register (AR).

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as 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

Use the following form of the LINK macro to invoke the ITTUWRIT service:

```
►►─label──┬─LINK──EP=ITTUWRIT──,MF=(E,parmarea)──────────────────────────────►◄
          └─LINKX──EP=ITTUWRIT──,MF=(E,parmarea)──┬──────────────────────┬──┘
                                                  └─,SF=(E,parmlist)─────┘
```

Note: As an alternative to using LINK or LINKX, callers in 31-bit AMODE can also:

1. Issue the MVS LOAD macro to load the ITTUWRIT service and obtain its entry point address.
2. Issue the CALL macro to call the service. Specify MF=(E,*your_parmlist*) on the call.

Parameters

The parameters are explained as follows:

label
The name on the macro invocation.

LINK
LINKX
Names the system service that is to be used for linkage.

EP=ITTUWRIT
Specifies the entry point name for the ITTUWRIT service.

,MF=(E,*parmarea*)
Specifies the address of the parameter list to be passed to ITTUWRIT. The parameter list consists of the following three addresses:

- The address of the token passed from ITTUINIT.
- The address of a fullword containing the size of the block of CTE entries.
- The address of the area containing the CTE entries.

,SF=(E,*parmlist*)
For use with LINKX when your program is reentrant. Before you call LINKX with this parameter, define *parmlist* using the LIST form of LINKX.

Return and Reason Codes

When the ITTUWRIT service returns control to your program, Register 15 contains a return code.

Decimal Return Code	Meaning and Action
00	**Meaning**: The ITTUWRIT request completed successfully. **Action**: None required.
16	**Meaning**: Warning. The ITTUWRIT request did not complete successfully. **Action**: Reissue ITTUWRIT.

ITZEVENT — Transaction Trace EVENT Record

Description

The ITZEVENT macro is used to build and record a transaction trace record. It optionally performs the query function to determine if the work unit should be traced.

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
Interrupt status:	Enabled forI/O and external interrupts
Locks:	No locks may be held
Control parameters:	Control parameters must be in the primary address space. The data pointed to by DATAADDR must reside in the caller's primary address space.

Programming Requirements

Any module that invokes this macro must include the macos CVT and IHAECVT.

To get the equate symbols for the return and reason codes, the caller should include the ITZYRETC macro.

Restrictions

None.

Input Register Information

Before issuing the ITZEVENT macro, the caller must ensure 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

Before issuing the ITZEVENT 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	Contains the reason code when 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	Contains the return code

© Copyright IBM Corp. 1988, 2002

When control returns to the caller, the ARs contain:

Register	Contents
0-1	Unpredictable (Used as a work register by the system)
2-13	Unchanged
14-15	Unpredictable (Used as a work register by the system)

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

Performance Implications

None.

Syntax

The ITZEVENT macro is written as follows:

---

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

COMPONENT=*component*	*component*: RS-type address or address in register (2) - (12)
,EVENTDESC=*eventdesc*	*eventdesc*: RS-type address or address in register (2) - (12)
,DATAFORMAT=**TT** ,DATAFORMAT=GTF	**Default:** DATAFORMAT=TT
,DATAADDR=*dataaddr*	*dataaddr*: RS-type address or address in register (2) - (12)
,DATALEN=*datalen*	*datalen*: RS-type address or address in register (2) - (12)
,DATAADDR=*dataaddr*	*dataaddr*: RS-type address or address in register (2) - (12)
,DATALEN=*datalen*	*datalen*: RS-type address or address in register (2) - (12)
,GTFID=*gtfid*	*gtfid*: RS-type address or address in register (2) - (12)
,GTFFID=*gtffid*	*gtffid*: RS-type address or address in register (2) - (12)
,FMTTYPE=**HEX** ,FMTTYPE=MODEL ,FMTTYPE=ROUTINE	**Default:** FMTTYPE=HEX
,FORMATRTN=*formatrtn*	*formatrtn*: RS-type address or address in register (2) - (12)

,FORMATRTN=*formatrtn*	*formatrtn*: RS-type address or address in register (2) - (12)
,FUNCTIONNAME=	*functionname* *functionname*: RS-type adderss or address in register (2) - (12)
,QUERY=YES ,QUERY=NO	**Default:** QUERY=YES
,MONTKN=*montkn*	*montkn*: RS-type address or address in register (2) - (12)
,TRACETKN=*tracetkn*	*tracetkn*: RS-type address or address in 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=(M,*list addr,*NOCHECK)	**Default**: MF=S *list addr*: RS-type address or register (1) - (12) ,MF=(E,*list addr*) ,MF=(E,*list addr,***COMPLETE**) ,MF=(E,*list addr,*NOCHECK) ,MF=(M,*list addr*) ,MF=(M,*list addr,***COMPLETE**)

Parameters

The parameters are explained as follows:

name
This is an optional symbol, starting in column 1, that is the name on the ITZEVENT macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

COMPONENT=*component*
This is a required input parameter that specifies the user component name used in formatting the standard transaction trace header.

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

,EVENTDESC=*eventdesc*
This is a required input parameter that specifies the event-related field used in formatting the standard transaction trace header.

Some examples might be START xxxxxxxx, END xxxxxxxx, ENTRYPTxxx, COMMIT, and ROLLBACK.

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

,DATAFORMAT=TT

,DATAFORMAT=GTF
This is an optional parameter that specifies the kind of data that follows the transaction trace header in the trace record. The default is DATAFORMAT=TT.

,DATAFORMAT=TT
The data recorded will contain transaction trace-related data.

,DATAFORMAT=GTF
This indicates that a GTF data record follows the standard transaction trace header. A pointer to the GTF record is passed along with the length.

,DATAADDR=*dataaddr*
When DATAFORMAT=TT is specified, this is an optional input parameter that can be used to specify the address and length of the data to be appended at the end of the transaction trace header. This is event-specific data set up by the user of this macro.

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

,DATALEN=*datalen*
When DATAADDR=dataaddr and DATAFORMAT=TT are specified, this is a required input parameter that specifies the length of the data to be appended at the end of the transaction trace header. This is event-specific data, set up by the user of this macro.

The maximum length of data may not exceed 1K. If a length greater than 1K is specified, data will be truncated to record 1K of data.

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

,DATAADDR=*dataaddr*
When DATAFORMAT=GTF is specified, this is a required input parameter that specifies the address and length of the GTF record to be appended at the end of the transaction trace header.

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

,DATALEN=*datalen*
When DATAFORMAT=GTF is specified, this is a required input parameter that specifies the length of the data to be appended at the end of the transaction trace header.

The maximum length of data may not exceed 1K. If a length greater than 1K is specified, data will be truncated to record 1K of data.

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

,GTFID=*gtfid*
When DATAFORMAT=GTF is specified, this is a required input parameter that 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.

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

,GTFFID=*gtffid*
When DATAFORMAT=GTF is specified, this is an optional input parameter that specifies the format appendage (fidname) that controls the formatting of the record. Formatting occurs when the trace output is processed by GTF trace.

The format appendage name is formed by appending the 2-digit GTFFID value to the names AMDUSER, HMDUSR, and IMDUSR. Assign GTFFID values as follows:

- X'00' - The record is to be dumped in hex.
- X'01' to X'50' - The record contains user format identifiers.

Note: If you omit the GTFFID parameter, the system supplies a default fidname of zero.

To code: Specify the RS-type address, or address in register (2)-(12), of a 1-character field.

,FMTTYPE=HEX
,FMTTYPE=MODEL
,FMTTYPE=ROUTINE

This is an optional parameter that specifies the IPCS format routine type for the user data. Refer to *z/OS MVS IPCS Customization* for details about the IPCS format.

The formatting can be in Hex, Model format, or from a Format routine. If a FORMATRTN is specified, FMTTYPE must be set to Routine or Model. The default is FMTTYPE=HEX.

,FMTTYPE=HEX
The data is displayed in Hex format.

.FMTTYPE=MODEL
The data is displayed in a format provided in a model format routine.

,FMTTYPE=ROUTINE
The data is displayed in a format provided in a user format routine.

,FORMATRTN=*formatrtn*

When FMTTYPE=MODEL is specified, this is a required parameter that specifies the name of the routine to be used for formatting the user data.

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

,FORMATRTN=*formatrn*

When FMTTYPE=ROUTINE is specified, this is a required parameter that specifies the name of the routine to be used for formatting the user data..

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

,FUNCTIONNAME=*functionname*

this is an optional input parameter that specifies the function (module|routine|label) that is making the trace entry. This value is displayed on the trace record formatted by IPCS.

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

,QUERY=YES
,QUERY=NO

This is an optional parameter that specifies whether query should be performed to determine if this work unit is to be traced.

Specifying QUERY=YES causes the same function to be performed as the ITZQUERY macro. If transaction trace is active for this work unit, a trace record is built and recorded. The default is QUERY=YES.

,QUERY=YES
Specifies that Query needs to be performed.

,QUERY=NO
Specifies that Query does not need to be performed.

The transaction trace token (TRACETKN) is a required input parameter. The TRACETKN is obtained by issuing a ITZQUERY macro just prior to issuing the ITZEVENT.

,MONTKN=*montkn*
When QUERY=YES is specified, an optional input parameter is specified and is used as the token to locate the current monitoring environment.

IBM recommends that MONTKN be specified for a monitoring environment to keep the query pathlength short and fast.

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

,TRACETKN=*tracetkn*
When QUERY=NO is specified, this is a required input parameter that specifies the transaction trace token returned from the previously performed query.

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

Note: Some existing components and products may have difficulty finding space in their data areas to hold a 32-byte transaction trace token. Apply APAR OW50696 to shorten the significant portion of the token to 8 bytes. With service for OW50696 applied, only the first 8 bytes of the 32-byte token will contain values other than binary zeros. Components that might not be able to exploit component trace due to the size of the 32-byte token may save just the first 8 bytes between uses, expanding it for use with transaction trace APIs by padding with binary zeros.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
This is 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
This 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
Specify 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 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 of the macro 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 Specify 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)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

This is 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 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, and use the execute form to call the service.

IBM recommends that you use and execute forms of ITZEVENT in the following order:

- Use ITZEVENT ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use ITZEVENT ...MF=(M,list-addr,NOCHECK) specifying the parameters that you want to change.
- Use ITZEVENT ...MF=(E,list-addr,NOCHECK) to execute the macro.

,list addr
This is 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
This is an optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the

parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

,COMPLETE
This specifies that the system is to check for required parameters and supply defaults for omitted optional parameters.

,NOCHECK
This 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 ITZEVENT macro returns control to your program:
- GPR 15 contains a return code.
- When the value in GPR 15 is not zero, GPR 0 contains a reason code.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. IBM support personnel may request the entire reason code, including the xxxx value.

Table 14. Return and Reason Codes for the ITZEVENT Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol:** ITZGOOD **Meaning:** Success - this work unit was traced. **Action:** None.
4	—	**Equate Symbol:** ITZNOTR **Meaning:** Work unit was not traced. **Action:** None. Reason code set to indicate the reason for not tracing.
4	xxxx0401	**Equate Symbol:** ITZNOTKN **Meaning:** Trace token was zero. **Action:** None.
4	xxxx0402	**Equate Symbol:** ITZNOACT **Meaning:** Transaction trace is not active. **Action:** None.
4	xxxx0403	**Equate Symbol:** ITZLATNT **Meaning:** Transaction trace is LATENT with LATENT=N set. **Action:** None.

Examples

```
ITZEVENT COMPONENT=COMP,
     EVENTDESC=DESC,
     DATAADDR=TTDATA,
     DATALEN=TTLEN
```

```
COMP   DC   CL8'COMP1  '
DESC   DC   CL16'START TRAN  '
TTDATA  DC  CL64
TTLEN  DC   F'64'
```

ITZQUERY — Transaction Trace Query

Description

The ITZQUERY macro is used to query whether a transaction or work unit should be traced.

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
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

Any module that invokes this macro must include the CVT and IHAECVT macros.

Restrictions

None.

Input Register Information

Before issuing the ITZQUERY 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

Before issuing the ITZQUERY macro, the caller does not have to place any information into any access register (AR).

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code
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

Specifying the MONTKN in a monitoring environment results in a faster query.

Syntax

The ITZQUERY macro is written as follows:

---

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

,MONTKN=*montkn* ,MONTKN=**0**	*montkn*: RS-type address Default: MONTKN=0
,TRACETKN=*tracetkn*	*tracetkn*: RS-type address
,TRACELVL=*tracelvl*	*tracelvl*: RS-type address
,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
This is an optional symbol, starting in column 1, that is the name on the ITZQUERY macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

,MONTKN=*montkn*
MONTKN=0
An optional input parameter that is the token used to locate the current monitoring environment.

It is recommended that MONTKN be specified for a monitoring environment to keep the query pathlength short and fast. The default is 0.

To code: Specify the RS-type address of a fullword field.

,TRACETKN=*tracetkn*
This is a required output parameter that specifies the transaction trace token returned from query.

To code: Specify the RS-type address of a 32-character field.

Note: Some existing components and products may have difficulty finding space in their data areas to hold a 32-byte transaction trace token. Apply APAR OW50696 to shorten the significant portion of the token to 8 bytes. With service for OW50696 applied, only the first 8 bytes of the 32-byte token will contain values other than binary zeros. Components that might not be able to exploit component trace due to the size of the 32-byte token may save just the first 8 bytes between uses, expanding it for use with transaction trace APIs by padding with binary zeros.

,TRACELVL=*tracelvl*
This is an optional output parameter that specifies the transaction trace indicator returned from query. A non-zero value implies that this work unit is eligible for tracing. A value of zero implies that this work unit is not eligible for tracing. In that case, the trace token is also set to zero.

To code: Specify the RS-type address of a one-byte field.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
This is 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
This 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
Specify 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 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 of the macro 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 Specify 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)**
,MF=(E,*list addr***,NOCHECK)**
,MF=(M,*list addr***)**
,MF=(M,*list addr***,COMPLETE)**
,MF=(M,*list addr***,NOCHECK)**

This is an optional input parameter that specifies the macro form.

Use MF=S to specify the standard form of the macro, which builds an inline parameter list and generates the macro invocation to transfer control to the service. MF=S is the default.

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter may be coded with the list form of the macro.

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

Use MF=M together with the list and execute forms of the macro for service routines that need to provide different options according to user-provided input. Use the list form to define a storage area; use the modify form to set the appropriate options; then use the execute form to call the service.

- Use ITZQUERY ...MF=(M,list-addr,COMPLETE) specifying appropriate parameters, including all required ones.
- Use ITZQUERY ...MF=(M,list-addr,NOCHECK), specifying the parameters that you want to change.
- Use ITZQUERY ...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 ITZQUERY macro returns control to your program, GPR 15 contains a return code.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.

Table 15. Return and Reason Codes for the ITZQUERY Macro

Equate Symbol	Meaning and Action	
0	—	**Equate Symbol:** ITZGOOD **Meaning:** Success. **Action:** Trace this work unit. Trace token is non-zero.
4	—	**Equate Symbol:** ITZNOTR **Meaning:** Work unit not to be traced. **Action:** Do not trace this work unit.

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 topic 251 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 topic 256 for the syntax of this request.
- REQUEST=READBLOCK, which reads a selected log block in a log stream. See topic 263 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 topic 268 for the syntax of this request.
- REQUEST=END, which ends a browse session. See topic 273 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.
ƀ	One or more blanks must precede IXGBRWSE.
IXGBRWSE	
ƀ	One or more blanks must follow IXGBRWSE.

REQUEST=START	
,STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or register (2) - (12).
,BROWSETOKEN=*browsetoken*	*browsetoken*: RS-type address or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,OLDEST	**Default**: OLDEST
,YOUNGEST	
,STARTBLOCKID=*startblockid*	*startblockid*: RS-type address or register (2) - (12).
,SEARCH=*search*	*search*: RS-type address or register (2) - (12).

GMT=YES GMT=NO	
VIEW=ACTIVE VIEW=ALL	**Default**: VIEW=ACTIVE
MODE=SYNC MODE=SYNCECB	**Default**: MODE=SYNC
,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

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.

,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).
,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 MODE=SYNCECB	**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=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.

,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
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.

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	
,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	**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.

,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
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.

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	**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
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.

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	**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)	**Default**: MF=S

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

Parameters for REQUEST=END

The parameters are explained as follows:

REQUEST=END
Requests that the browse session be ended.

,STREAMTOKEN=*streamtoken*
Specifies the name or address (using a register) of a required 16-byte input field containing the token for the log stream that you want to search. The stream token is returned by the IXGCONN service at connection to the log stream.

,BROWSETOKEN=*browsetoken*
Specifies the name or address (using a register) of a required 4-byte input field containing the identifier for the browse session which was returned from the IXGBRWSE REQUEST=START request.

,ANSAREA=*ansarea*
Specifies the name (or address in a register) of an answer area containing information about this request. The answer area must be at least 40 bytes. To map this information, use the IXGANSAA macro.

,ANSLEN=*anslen*
Specifies the name (or address in a register) of the 4-byte field containing the answer area length. The length of the answer area must be at least 40 bytes and must be the same length as the field specified in ANSAREA.

To ascertain the optimal answer area length, look at the ANSAA_PREFERRED_SIZE field of the IXGANSAA macro.

,MODE=SYNC
,MODE=SYNCECB
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.

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 16. 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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0403	**Equate Symbol**: IxgRsnCodeWarningGap **Explanation**: Environment error. The request completed successfully, but the data requested was unreadable. The next readable data in the log stream in the specified direction is returned. This condition could be caused by either an I/O error while attempting to read a log data set or a log data set deleted without using the 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.
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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0417	**Equate Symbol**: IxgRsnCodeMultiblockErrorWarning **Explanation**: Environment error. Returned for READCURSOR requests with MULTIBLOCK=YES specified only. A log block has been returned, but an error condition was encountered while attempting to read more data. This may be issued when some log block data is returned and an end of the log stream (eof) is reached. **Action**: The action necessary is completely up to the application, depending on how critical your data is. You can do one of the following: • Accept this condition and continue reading. • Stop processing the log all together. • Attempt to get the problem rectified, if possible, and then attempt to re-read the log data.
08	xxxx0801	**Equate Symbol**: IxgRsnCodeBadParmlist **Explanation**: Program error. The parameter list could not be accessed. **Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx0802	**Equate Symbol**: IxgRsnCodeXESError **Explanation**: System error. A severe cross-system extended services (XES) error has occurred. **Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.
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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 16. Return and Reason Codes for the IXGBRWSE 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	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.

Table 16. Return and Reason Codes for the IXGBRWSE 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**: 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.
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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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	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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0848	**Equate Symbol**: IxgRsnCodeEndReached **Explanation**: Environment error. The request failed and no log data is returned. For a READCURSOR request, the end of the log stream has been reached in the direction of the read. If the SEARCH parameter was specified, 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.
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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx084B	**Equate Symbol**: IxgRsncodeLossOfDataGap **Explanation**: Environment error. The requested log data referenced a section of the log stream where log data is permanently missing. This condition occurs when a system or coupling facility is in recovery due to a failure, but not all of the log data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
08	xxxx084D	**Equate Symbol**: 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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0861	**Equate Symbol**: IxgRsnCodeRebuildInProgress **Explanation**: Environment error. No requests can be processed for this log stream because a coupling facility structure re-build is in progress for the structure associated with this log stream. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0862	**Equate Symbol**: IxgRsnCodeXESPurge **Explanation**: Environment error. An cross-system extended services (XES) request has been purged due to re-build processing. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
08	xxxx0863	**Equate Symbol**: IxgRsnCodeStructureFailed **Explanation**: Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action**: Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the re-build completed successfully. Re-issue the request. • The re-build failed and the log stream is not available.
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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0890	**Equate Symbol**: IxgRsnCodeAddrSpaceNotAvail **Explanation**: System error. The system logger address space failed and is not available. **Action**: Do not issue system logger requests.
08	xxxx0891	**Equate Symbol**: IxgRsnCodeAddrSpaceInitializing **Explanation**: System error. The system logger address space is not available because it is IPLing. **Action**: Listen for ENF signal 48, which will indicate when the system logger address space is available. Re-connect to the log stream, then re-issue this request. You can also listen for ENF signal 48, which will indicate if the system logger address space will not be available for the life of the IPL. In that case, do not issue system logger services.
08	xxxx08D0	**Equate Symbol**: IxgRsnCodeProblemState **Explanation**: Environment error. The request was rejected because of one of the following: • The request was issued in SRB mode while the requestor was in problem program state. • The SYNCEXIT parameter was specified while the requestor's PSW key was in problem program key. **Action**: Change the invoking environment to supervisor state.
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.

Table 16. Return and Reason Codes for the IXGBRWSE Macro (continued)

Return Code	Reason Code	Meaning and Action
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.

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.

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.

© 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 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).
,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

,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.

,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.

,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:
  - IMPORTCONNECT
  - LSVERSION

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 17. 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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0407	**Equate Symbol**: IxgRsnCodeConnPossibleLossOfData **Explanation**: Environment error. The request was successful, but there may be log blocks permanently missing between this log block and the one previously returned. This condition occurs when a system or coupling facility fails and not all of the data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
04	xxxx0408	**Equate Symbol**: IxgRsnCodeDsDirectoryFullWarning **Explanation**: Environment error. The request was successful, but the DASD data set directory for the log stream is now full. System logger cannot offload any further data to DASD. System logger will continue to process IXGWRITE requests only until the coupling facility structure space for this log stream is full. **Action**: Either delete data from the log stream to free up space in the data set directory or disconnect from the log stream.
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.

Table 17. Return and Reason Codes for the IXGCONN 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	xxxx0806	**Equate Symbol**: IxgRsnCodeBadStmToken **Explanation**: Program error. The stream token was not valid. **Action**: Make sure that the stream token specified is valid.
08	xxxx0808	**Equate Symbol**: IxgRsnCodeEIOError **Explanation**: System error. A severe log data set I/O error has occurred. **Action**: Contact the IBM Support Center. Provide the return and reason code.
08	xxxx080A	**Equate Symbol**: IxgRsnCodeRequestLocked **Explanation**: Program error. The program issuing the request is holding a lock. **Action**: Ensure that the program issuing the request is not holding a lock.
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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 17. Return and Reason Codes for the IXGCONN 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	xxxx0819	**Equate Symbol**: IxgRsnCodeSRBMode **Explanation**: Program error. The calling program is in SRB mode, but task mode is the required dispatchable unit mode for this system logger service. **Action**: Make sure the calling program is in task mode.
08	xxxx081A	**Equate Symbol**: IxgRsnCodeMaxStreamConn **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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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	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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0843	**Equate Symbol**: IxgRsnCodeXcdsReformat **Explanation**: Program error. A couple data set record is not valid. **Action**: Format the system logger couple data set again.
08	xxxx084C	**Equate Symbol**: IxgRsnCodeRMAlreadyConnected **Explanation**: Program error. The resource manager is trying to connect to a log stream that it is already connected to. Only one connection specifying RMNAME can be active for a log stream. **Action**: Correct the program so that it does not try to reconnect to the log stream.
08	xxxx084E	**Equate Symbol**: IXGRSNCODESTRSACETOOSMALL **Explanation**: Environment error. Structure resources are not available to satisfy the request. All structure resources are allocated as system logger control resources. This condition occurs when the structure resources are consumed by the logstreams connections. **Action**: Increase the size of the structure in the CFRM policy or use SETXCF ALTER support to dynamically increase the size of the structure.
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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0851	**Equate Symbol**: IXGRSNCODEBADCFLEVEL **Explanation**: Environment error. The connect request was rejected. The operational level of the coupling facility is not sufficient to support logger functions. **Action**: Ensure that the coupling facility operational level for logger structures is at the required level. See *z/OS MVS Setting Up a Sysplex*.
08	xxxx0853	**Equate Symbol**: IxgRsnCodeNoCF **Explanation**: Environment error. The connect request was rejected. System logger could not allocate coupling facility structure space because no suitable coupling facility was available. **Action**: Check accompanying message IXG206I for a list of the coupling facilities where space allocation was attempted and the reason why each attempt failed.
08	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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx08D6	**Equate Symbol**: IXGRsnCodeConnTypeNotAllowed **Explanation**: Environment error. One of the following occurred: • The connect request specified IMPORTCONNECT=YES, but there is already an active write connection (AUTH=WRITE IMPORTCONNECT=NO) in the sysplex. You cannot have an import connection and a write connection to the same log stream. • The connect request specified AUTH=WRITE IMPORTCONNECT=NO, but there is already an active import connection (IMPORTCONNECT=YES) for the log stream. You cannot have an import connection and a write connection to the same log stream. You can only have one import connection to a log stream. You may have multiple write connections, as long as there is no import connection against a log stream. **Action**: Correct your program and retry the request.
08	xxxx08E2	**Equate Symbol**: IxgRsncodeDasdOnlyConnected **Explanation**: Environment error 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.

Table 17. Return and Reason Codes for the IXGCONN Macro (continued)

Return Code	Reason Code	Meaning and Action
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
```

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
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	**Default**: MODE=SYNC
,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
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.

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

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 18. 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.

Table 18. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
04	xxxx040E	**Equate Symbol**: IxgRsnCodeRMBadGap **Explanation**: Environment error. The IXGDELET request failed because the requested log data was unreadable. This problem is caused by either an I/O error while attempting to read a DASD log data set or a log data set was deleted using an interface other than IXGDELET. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request. **Action**: System logger returns the block identifier of the first readable log block (in the direction of youngest data) in the ANSAA_GAPS_NEXT_BLKID field of the answer area mapped by IXGANSAA. If appropriate, reissue the IXGDELET request using this block identifier.

Table 18. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx040F	**Equate Symbol**: IxgRsnCodeRMEOFGap **Explanation**: Environment error. While processing the IXGDELET request, system logger prematurely reached the end or beginning of the log stream. The portion of the log stream from the requested log data to either the beginning or end of the log stream was unreadable. This problem is caused by either an I/O error while attempting to read a DASD log data set or a log data set was deleted using an interface other than IXGDELET. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request. **Action**: The action you take depends on whether your application can tolerate any loss of data. You can either: • Accept the loss of data and continue processing this log stream. • Stop using this log stream. • Correct the problem and re-issue the request.
04	xxxx0410	**Equate Symbol**: IxgRsnCodeRMLossOfDataGap **Explanation**: Environment error. The log data you tried to delete is in a section of the log stream where data is permanently missing. This condition occurs when a system or coupling facility is in recovery from a failure and not all the log data could be recovered. This reason code is issued only when a resource manager exit overrides the block identifier specified on the IXGDELET request. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. If your application can tolerate data loss, you can continue using the log stream.
04	xxxx0411	**Equate Symbol**: IxgRsnCodeRMAbended **Explanation**: Program error. The resource manager abended and percolated to the system logger recovery environment. The IXGDELET request was not processed. **Action**: Look for and correct the problem in your resource manager program or reissue the delete request, specifying FORCE=YES.
04	xxxx0412	**Equate Symbol**: IxgRsnCodeRMDisabled **Explanation**: Environment error. The log stream is identified as being managed by a resource manager (RMNAME is specified in the 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.

Table 18. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 18. Return and Reason Codes for the IXGDELET Macro (continued)

Return Code	Reason Code	Meaning and Action
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 18. 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 18. 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 18. 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 18. 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*: RS-type address or address in register (2) - (12).
,BUFFALET=**0,**	**Default**: BUFFALET=0,
,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.

,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 19. Return and Reason Codes for the IXGIMPRT Macro

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

Table 19. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 19. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 19. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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	**IxgRsnCodeBadAnslen -** **Explanation**: Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro. **Action**: Reissue the request, specifying an answer area of the required size.
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.
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

Table 19. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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).
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.

Table 19. Return and Reason Codes for the IXGIMPRT 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 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.
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.

Table 19. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 19. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 19. Return and Reason Codes for the IXGIMPRT Macro (continued)

Return Code	Reason Code	Meaning and Action
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
```

IXGINVNT — Managing the LOGR Inventory Couple Data Set

Description

The LOGR policy tracks all data associated with log streams, such as log stream characteristics, coupling facility structures associated with log streams, and the systems connected to each log stream.

Use the IXGINVNT macro to manage the LOGR policy by:
- Defining, updating or deleting entries for log streams in the LOGR policy.
- Defining or deleting entries for coupling facility structures in the LOGR policy.

The three requests for the macro are:
- IXGINVNT REQUEST=DEFINE, which defines an entry in the LOGR policy. There are two types of DEFINE requests:
  - TYPE=LOGSTREAM defines an entry for a log stream. See topic 347 for the syntax of this request.
  - TYPE=STRUCTURE defines an entry for a system logger coupling facility structure. See topic 361 for the syntax of this request.
- IXGINVNT REQUEST=UPDATE, which updates a log stream entry in the LOGR policy. See topic 365 for the syntax of this request.
- IXGINVNT REQUEST=DELETE, which deletes a log stream or structure entry from the LOGR policy. See topic 379 for the syntax of this request.

For information on using the system logger services and the LOGR policy, 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:	Control parameters must be in the primary address space.

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.

© Copyright IBM Corp. 1988, 2002

- The caller cannot have an EUT FRR established.
- You can only use the IXGINVNT REQUEST=DELETE TYPE=LOGSTREAM request to delete a log stream entry from the LOGR policy if there are no connections (active or failed) to the log stream.
- For most parameters on the IXGINVNT REQUEST=UPDATE request, there must be no connections (active or failed) to the log stream being updated. The AUTODELETE and RETPD parameters are the exception, as noted in the parameter descriptions.
- Restrictions for DASD-only log stream definitions:
  - A DASD-only log stream is single-system in scope. This means that only one system at a time may connect to a DASD-only log stream. You **can** have multiple connections from one system or multiple systems connecting in sequence.
  - A DASD-only log stream is not associated with a coupling facility structure.
  - If the requested function is to update the attributes of a DASD-only log stream, then the following parameters are not allowed:
    - STG_DUPLEX
    - DUPLEXMODE
    - LOGGERDUPLEX

    Use of staging data sets is automatic rather than optional for a DASD-only log stream.
  - A DASD-only log stream can be upgraded to a coupling facility log stream by specifying STRUCTNAME on the IXGINVNT REQUEST=UPDATE TYPE=LOGSTREAM request

    Conversely, a coupling facility log stream cannot be changed to DASD-only nor can STRUCTNAME be specified on an update request issued against a coupling facility log stream.
- If the Security Authorization Facility (SAF) is available, the system performs SAF authorization checks on all IXGINVNT requests.

  For log stream entries, you must have the following authorization:
  - To define, delete, or update a log stream entry, the caller must have alter access to RESOURCE(*log_stream_name*) in SAF class CLASS(LOGSTREAM)
  - If you specify the STRUCTNAME parameter on a DEFINE request for a log stream entry, the caller must also have update access authority to the coupling facility structure, RESOURCE(IXLSTR.*structure_name*) in SAF class CLASS(FACILITY)
  - If you use the LIKE parameter to model your definition after another log stream on a DEFINE request for a log stream entry, you must also have update access to the RESOURCE(IXLSTR.*like_structure_name*) in class CLASS(FACILITY).

  To define or delete a structure entry in the LOGR policy, the caller must have alter access to RESOURCE(MVSADMIN.LOGR) in SAF class CLASS(FACILITY).

  If SAF is not available or if there is no CLASS(LOGSTRM) or CLASS(FACILITY) class defined for the log stream or structure, no security checking is performed.
- 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 IXGINVNT 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.

REQUEST=DEFINE TYPE=LOGSTREAM Option of IXGINVNT

The IXGINVNT macro with the DEFINE TYPE=LOGSTREAM parameters defines a log stream or coupling facility structure entry in the LOGR policy.

Syntax for REQUEST=DEFINE TYPE=LOGSTREAM

The standard form of the IXGINVNT REQUEST=DEFINE TYPE=LOGSTREAM macro is written as follows:

---

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

REQUEST=DEFINE

,TYPE=LOGSTREAM

,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,STREAMNAME=*streamname*	*streamname*: RS-type address or register (2) - (12).
,STRUCTNAME=*structname*	*structname*: RS-type address or register (2) - (12). **Default**: NO_STRUCTNAME
,DASDONLY=NO ,DASDONLY=YES	**Default**: DASDONLY=NO
,MAXBUFSIZE=*maxbufsize*	*maxbufsize*: RS-type address or register (2) - (12).
,RMNAME=*rmname*	*rmname*: RS-type address or register (2) - (12).
,DESCRIPTION=*description*	*description*: RS-type address or register (2) - (12). **Default**: NO_DESCRIPTION
,LOGGERDUPLEX=UNCOND ,LOGGERDUPLEX= COND	**Default**: LOGGERDUPLEX=UNCOND
,STG_DUPLEX=NO ,STG_DUPLEX=YES	**Default**: STG_DUPLEX=NO
,DUPLEXMODE=COND ,DUPLEXMODE=UNCOND	**Default**: DUPLEXMODE=COND
,STG_MGMTCLAS=*stg_mgmtclas*	*stg_mgmtclas*: RS-type address or register (2) - (12). **Default**: NO_STG_MGMTCLAS
,STG_DATACLAS=*stg_dataclas*	*stg_dataclas*: RS-type address or register (2) - (12). **Default**: NO_STG_DATACLAS
,STG_STORCLAS=*stg_storclas*	*stg_storclas*: RS-type address or register (2) - (12). **Default**: NO_STG_STORCLAS
,STG_SIZE=*stg_size*	*stg_size*: RS-type address or register (2) - (12). **Default**: Size defined in SMS data class or by dynamic allocation
,LS_MGMTCLAS=*ls_mgmtclas*	*ls_mgmtclas*: RS-type address or register (2) - (12). **Default**: NO_LS_MGMTCLAS
,LS_DATACLAS=*ls_dataclas*	*ls_dataclas*: RS-type address or register (2) - (12). **Default**: NO_LS_DATACLAS
,LS_STORCLAS=*ls_storclas*	*ls_storclas*: RS-type address or register (2) - (12). **Default**: NO_LS_STORCLAS
,LS_SIZE=*ls_size*	*ls_size*: RS-type address or register (2) - (12). **Default**: Size defined in SMS data class or by dynamic allocation
,RETPD=*retpd*	*retpd*: RS-type address or register (2) - (12). **Default**: NO_RETPD

,AUTODELETE=NO ,AUTODELETE=YES	**Default**: AUTODELETE=NO
,HLQ=*hlq*	*hlq*: RS-type address or register (2) - (12). **Default**: NO_HLQ
,EHLQ=*ehlq*	*ehlq*: RS-type address or register (2) - (12). **Default**: NO_EHLQ
,LOWOFFLOAD=*lowoffload*	*lowoffload*: RS-type address or register (2) - (12). **Default**: LOWOFFLOAD=0
,HIGHOFFLOAD=*highoffload*	*highoffload*: RS-type address or register (2) - (12). **Default**: HIGHOFFLOAD=80
,LIKE=*like_streamname*	*like*: RS-type address or register (2) - (12). **Default**: NO_LIKE
,MODEL=NO ,MODEL=YES	**Default**: MODEL=NO
,DIAG=NO ,DIAG=YES	**Default**: DIAG=NO
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=0 ,PLISTVER=1 ,PLISTVER=2 ,PLISTVER=3	**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=DEFINE,TYPE=LOGSTREAM

The parameters are explained as follows:

REQUEST=DEFINE
Requests that an entry for a log stream or coupling facility structure be defined in the LOGR policy.

,TYPE=LOGSTREAM
Indicates that the entry to be defined in the LOGR policy is a log stream entry.

,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.

,STREAMNAME=*streamname*
Specifies the name (or address in a register) of the 26-byte input field containing the name of the log stream that you want to define in the LOGR policy.

The stream name must be 26 characters, padded on the right with blanks if necessary. The name can be made up of one or more segments separated by periods, up to the maximum length of 26 characters. The following rules apply:

- Each segment may contain up to eight numeric, alphabetic, or national ($, #, or @) characters.
- The first character of each segment must be an alphabetic or national character.
- Each segment must be separated by periods, which you must count as characters.

STREAMNAME is required with the TYPE=LOGSTREAM parameter.

,STRUCTNAME=*structname*
With TYPE=LOGSTREAM, specifies the name (or address in a register) of a 16-byte input field that contains the name of the coupling facility structure associated with the coupling facility log stream being defined. The structure specified is a list structure defined in the CFRM policy. All of this log stream's log blocks will be written to this structure before being written to DASD.

For a coupling facility log stream, you must define STRUCTNAME in the log stream definition in the LOGR policy via this parameter or the STRUCTNAME defined for the log stream referenced by the LIKE parameter before you can connect to the log stream.

The following rules apply for the structname:

- It may contain numeric, alphabetic, or national ($, #, or @) characters, or an underscore(_), padded on the right with blanks if necessary.
- The first character must be an alphabetic character.

For a DASD-only log stream, omit the STRUCTNAME parameter, since there is no coupling facility associated with the log stream.

If NO_STRUCTNAME is specified for STRUCTNAME, the macro will be invoked as if STRUCTNAME was not specified.

,DASDONLY=NO
,DASDONLY=YES
Specifies whether the log stream being defined is a coupling facility or a DASD-only log stream.

If you specify DASDONLY=NO, which is the default, the log stream is defined as a coupling facility log stream.

If you specify DASDONLY=YES the log stream is defined as a DASD-only log stream and does not use the coupling facility for log data.

With DASDONLY=NO, you can also specify STG_DUPLEX, DUPLEXMODE and LOGGERDUPLEX parameters to select a method of duplexing for a coupling facility log stream.

,MAXBUFSIZE=*maxbufsize*

Specifies the name (or address in a register) of a 4-byte input field that contains the size, in bytes, of the largest log block that can be written to the DASD-only log stream being defined in this request.

The value for MAXBUFSIZE must be between 1 and 65,532 bytes. The default is 65,532 bytes.

This parameter is valid only with DASDONLY=YES.

,RMNAME=*rmname*

Specifies the name (or address in a register) of the 8-byte input field containing the name of the recovery resource manager program associated with the log stream. RNAME must be 8 alphanumeric or national ($,#,or @) characters, padded on the right with blanks if necessary.

You must define RMNAME in the LOGR policy before the resource manager can connect to the log stream.

If you specify RMNAME to associate a resource manager with a log stream in the LOGR policy, the resource manager specified must subsequently connect to the log stream. If the resource manager does not connect to that log stream, system logger will not process any IXGDELET requests to delete log data. This is so that the resource manager will not miss any delete requests issued against the log stream.

,DESCRIPTION=NO_DESCRIPTION
DESCRIPTION=*description*

Specifies the name (or address in a register) of the 16 character input field containing user defined data describing the log stream.

DESCRIPTION must be 16 alphanumeric or national ($,#,@) characters, underscore (_) or period (.), padded on the right with blanks if necessary.

If you specify DESCRIPTION=NO_DESCRIPTION, which is the default, or a field of zeros, the macro is invoked as if the DESCRIPTION parameter was not specified.

,LOGGERDUPLEX=UNCOND
,LOGGERDUPLEX=COND

An optional input parameter that specifies whether Logger continues to provide its own log data duplexing, or, conditionally, not provide its own duplexing based on an alternative duplexing configuration that provides an equivalent or better recoverability of the log data.

The active primary TYPE=LOGR couple data set in the sysplex must be formatted at z/OS Release 2 or higher to specify this keyword. Otherwise, the request fails with a return code 8, reason code 0839.

This option is not allowed for DASDONLY log streams. Logger continues to unconditionally provide its own duplexing of DASDONLY log stream data.

Refer to Logger and CF Duplexing Combinations and System Logger Recovery in *z/OS MVS Setting Up a Sysplex* for additional considerations on using the LOGGERDUPLEX parameter.

LOGGERDUPLEX=UNCOND, which is the default, indicates that Logger should provide its own duplexing of the log data regardless of any other duplexing (such as structure system-managed duplexing rebuild) that may be occurring.

LOGGERDUPLEX=COND indicates that Logger should provide its own duplexing of the log data unless the log stream is in an alternative duplexing configuration that provides an equivalent or better recoverability of the log data. For example, Logger does not provide its own duplexing of the log data in the following configuration:

- when the log stream is in a non-volatile CF list structure that is handled by system-managed duplexing rebuild (duplex-mode),
- there is a failure-independent relationship between the two structure instances, and
- there is a failure-independent connection between connecting system and composite structure view.

,STG_DUPLEX=NO
,STG_DUPLEX=YES

Specifies whether the log stream data for a coupling facility log stream should be duplexed in DASD staging data sets.

If you specify STG_DUPLEX=NO, which is the default, log data for a coupling facility log stream will not be duplexed in staging data sets.

If you specify STG_DUPLEX=YES, the log data for a coupling facility log stream will be duplexed in staging data sets if the conditions defined by the DUPLEXMODE parameter are fulfilled.

You can use the DUPLEXMODE parameter with STG_DUPLEX and with LOGGERDUPLEX to specify the type of duplexing desired and whether you want conditional or unconditional duplexing by Logger.

STG_DUPLEX is only valid when you specify or default to DASDONLY=NO to define a coupling facility log stream.

,DUPLEXMODE=COND
DUPLEXMODE=UNCOND

Specifies the conditions under which the coupling facility log data for a coupling facility log stream should be duplexed in DASD staging data sets.

If you specify DUPLEXMODE=COND, which is the default, the coupling facility log data is duplexed in staging data sets only if a system's connection to the coupling facility log stream contains a single point of failure and is therefore vulnerable to permanent log data loss:

- A connection to a log stream contains a single point of failure if the coupling facility is volatile and/or resides on the same CPC as the MVS system connecting to it. The coupling facility log data for the system connection containing the single point of failure is duplexed to staging data sets.
- A connection to a log stream is failure-independent when the coupling facility for the log stream is non-volatile and resides on a different central processor complex (CPC) than the MVS system connecting to it. The coupling facility log data for that system connection will not be duplexed to staging data sets.

If you specify DUPLEXMODE=UNCOND, the log data for the coupling facility log stream will be duplexed in staging data sets, unconditionally, even if the connection is failure independent.

You can use the DUPLEXMODE parameter with STG_DUPLEX and with LOGGERDUPLEX to specify the type of duplexing desired and whether you want conditional or unconditional duplexing by Logger.

Refer to Selecting a Method of Duplexing Coupling Facility Log Data and System Logger Recovery in *z/OS MVS Setting Up a Sysplex* for complete information on using staging data sets to duplex coupling facility log data.

DUPLEXMODE is valid only when STG_DUPLEX=YES has been specified for a coupling facility log stream.

,STG_DATACLAS=*NO_STG_DATACLAS*
,STG_DATACLAS=*stg_dataclas*
Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS data class that will be used for allocation of the DASD staging data set for this log stream.

The data class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

If you specify NO_STG_DATACLAS, which is the default, or a field of zeros, the class is assigned by standard SMS processing. See *z/OS DFSMS: Using Data Sets* for more information about SMS.

An SMS value specified on the STG_DATACLAS parameter, including NO_STG_DATACLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

STG_DATACLAS is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,STG_MGMTCLAS=*NO_STG_MGMTCLAS*
,STG_MGMTCLAS=*stg_mgmtclas*
Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS management class that will be used for allocation of the DASD staging data set for this log stream.

The management class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

If you specify NO_STG_MGMTCLAS, which is the default, or a field of zeros, the class is assigned by standard SMS processing. See *z/OS DFSMS: Using Data Sets* for more information about SMS.

An SMS value specified on the STG_MGMTCLAS parameter, including NO_STG_MGMTCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

STG_MGMTCLAS is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,STG_STORCLAS=*NO_STG_STORCLAS*
,STG_STORCLAS=*stg_storclas*
Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS storage class that will be used for allocation of the DASD staging data set for this log stream.

The storage class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

If you specify NO_STG_STORCLAS, which is the default, or a field of zeros, the class is assigned by standard SMS processing. See *z/OS DFSMS: Using Data Sets* for more information about SMS.

An SMS value specified on the STG_STORCLAS parameter, including NO_STG_STORCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

STG_STORCLAS is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,STG_SIZE=*stg_size*

Specifies the name (or address in a register) of a 4-byte input field containing the size, in 4K blocks, of the DASD staging data set for the log stream being defined.

If you omit STG_SIZE, for a **coupling facility log stream**, system logger does one of the following, in the order listed, to allocate space for staging data sets:

- Uses the STG_SIZE of the log stream specified on the LIKE parameter, if specified.
- Uses the maximum coupling facility structure size for the structure to which the log stream is defined. This value is obtained from the value defined on the SIZE parameter for the structure in the CFRM policy.

If you omit STG_SIZE for a **DASD-only log stream**, system logger does one of the following, in the order listed, to allocate space for staging data sets:

- Uses the STG_SIZE of the log stream specified on the LIKE parameter, if specified.
- Uses the size defined in the SMS data class for the staging data sets.
- Uses dynamic allocation rules for allocating data sets, if SMS is not available.

Note that if both the STG_DATACLAS and STG_SIZE are specified, the value for STG_SIZE overrides the space allocation attributes for the data class specified on the STG_DATACLAS value.

STG_SIZE is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,LS_DATACLAS=*NO_LS_DATACLAS*
,LS_DATACLAS=*ls_dataclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS data class that will be used for allocation of the DASD log data set for this log stream.

The data class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

If you specify NO_LS_DATACLAS, which is the default, or a field of zeros, the class is assigned by standard SMS processing. See *z/OS DFSMS: Using Data Sets* for more information about SMS.

An SMS value specified on the LS_DATACLAS parameter, including NO_LS_DATACLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

,LS_MGMTCLAS=*NO_LS_MGMTCLAS*

,LS_MGMTCLAS=*ls_mgmtclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS management class that will be used for allocation of the DASD log data set for this log stream.

The management class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

If you specify NO_LS_MGMTCLAS, which is the default, or a field of zeros, the class is assigned by standard SMS processing. See *z/OS DFSMS: Using Data Sets* for more information about SMS.

An SMS value specified on the LS_MGMTCLAS parameter, including NO_LS_MGMTCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

,LS_STORCLAS=*NO_LS_STORCLAS*
,LS_STORCLAS=*ls_storclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS storage class that will be used for allocation of the DASD log data set for this log stream.

The storage class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

If you specify NO_LS_MGMTCLAS, which is the default, or a field of zeros, the class is assigned by standard SMS processing. See *z/OS DFSMS: Using Data Sets* for more information about SMS.

An SMS value specified on the LS_MGMTCLAS parameter, including NO_LS_MGMTCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

,LS_SIZE=*ls_size*

Specifies the name (or address in a register) of a 4-byte input field containing the size, in 4K blocks, of the DASD log data set for the log stream being defined.

If you omit LS_SIZE, or specify a field of zeros, the value is assigned by standard SMS processing. See *z/OS DFSMS: Using Data Sets* for more information.

Note that a value specified on the LS_SIZE parameter overrides the space allocation attributes for the data class specified on the LS_DATACLAS parameter.

,AUTODELETE=*NO*
,AUTODELETE=YES

Specifies when system logger physically deletes log data.

If you specify AUTODELETE=NO, which is the default, system logger physically deletes an entire log data set only when **both** of the following are true:

- Data is marked for deletion by a system logger application using the IXGDELET service.
- The retention period for all the data in the log data set expires.

You must specify the RETPD parameter with AUTODELETE=NO.

If you specify AUTODELETE=YES, system logger automatically physically deletes log data whenever data is either marked for deletion (using the IXGDELET service or an archiving procedure) or the retention period for all the log data in a data set has expired.

Be careful when using AUTODELETE=YES if the system logger application manages log data deletion using the IXGDELET service. With AUTODELETE=YES, system logger may delete data that the application expects to be accessible.

The LOGR couple data set must be formatted at the OS/390 Release 3 level or above to use this keyword.

RETPD=*0*
RETPD=*retpd*
Specifies the name (or address in a register) of a 4-byte input field containing the number of days of the retention period for log data in the log stream. The retention period begins when data is written to the log stream. Once the retention period for an entire log data set has expired, the data set is eligible for physical deletion. The point at which system logger physically deletes the data depends on what you have specified on the AUTODELETE parameter. System logger will not process a retention period or delete data on behalf of log streams that are not connected to or being written to by an application.

The value specified for RETPD must be between 0 and 65,536.

The LOGR couple data set must be formatted at the OS/390 Release 3 level or above to use this keyword.

,HLQ=*NO_HLQ*
,HLQ=*hlq*
Specifies the name (or address in a register) of an 8-byte input field containing the high-level qualifier for both the log stream data set name and the staging data set name.

The high-level qualifier must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

If you specify an explicit value for HLQ, this value overrides a high-level qualifier for the log stream specified on the LIKE parameter.

If you do not specify a high-level qualifier, or if you specify HLQ=NO_HLQ, the log stream being defined will have a high-level qualifier of IXGLOGR. If you also specified the LIKE parameter, it will have the high-level qualifier of the log stream specified on the LIKE parameter.

HLQ and EHLQ are mutually exclusive and cannot be specified for the same log stream definition.

If the name specified for the HLQ parameter refers to a field that contains X'00', then the macro will be invoked as if NO_HLQ had been specified. However, specifying HLQ=NO_HLQ and EHLQ=*ehlq* on the same request results in an error. When HLQ=NO_HLQ is specified, then the resulting high-level qualifier will be determined by the EHLQ value from the LIKE log stream or using a default value.

,EHLQ=*NO_EHLQ*

,EHLQ=*ehlq*

Specifies the name (or address in a register) of a 33-byte input field containing the extended high-level qualifier for both the log stream data set name and the staging data set name.

Syntax requirements for the extended high-level qualifier are as follows:

- The extended high-level qualifier must be 33 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary.
- The value can be made up of one or more qualifiers (each 1 to 8 characters) separated by periods, up to the maximum length of 33 characters.
- Each qualifier must contain up to eight alphabetic, national, or numeric characters. Lowercase alphabetic characters will be folded to uppercase.
- The first character of each qualifier must be an alphabetic or national character.
- Each qualifier must be separated by a period, which you must count as a character.
- The resulting length of concatenating the significant characters from the EHLQ value with the STREAMNAME value (including the period delimiter) cannot exceed 35 characters.

EHLQ and HLQ are mutually exclusive and cannot be specified for the same log stream definition.

When the EHLQ parameter is not explicitly specified on the request, the resulting high-level qualifier to be used for the log stream data sets will be based on whether the HLQ or LIKE parameters are specified. If the HLQ parameter is specified, then that value will be used for the log stream data sets. When no high-level qualifier is explicitly specified on the DEFINE LOGSTREAM request, but the LIKE parameter is specified, the the high-level qualifier value being used in the referenced log stream will be used for the newly defined log stream. If the EHLQ, HLQ, and LIKE parameters are not specified, then the default value “IXGLOGR” will be used.

If the name specified for the EHLQ parameter refers to a field that contains X'00', then the macro will be invoked as if NO_EHLQ had been specified. However, specifying EHLQ=NO_EHLQ and HLQ=*hlq* on the same request results in an error. When EHLQ=NO_EHLQ is specified, then the resulting high-level qualifier will be determined by the HLQ value from the LIKE log stream or using a default value.

The active primary TYPE=LOGR couple data set must be formatted at a z/OS release 1.2 or higher level in order to specify the EHLQ keyword. Otherwise, the request will fail with return code 8, reason code X'0839'.

,HIGHOFFLOAD=*80*
,HIGHOFFLOAD=*highoffload*

Specifies the name (or address in a register) of an 4-byte input field containing the percent value you want to use as the high offload threshold for the coupling facility structure associated with this log stream. When the coupling facility is filled to the high offload threshold percentage or beyond, system logger begins offloading data from the coupling facility to the DASD log stream data sets.

If you specify HIGHOFFLOAD=80, which is the default, HIGHOFFLOAD=0, or omit the HIGHOFFLOAD parameter, system logger uses the 80% usage mark as the high offload threshold where offloading starts.

IBM recommends that you do not define your HIGHOFFLOAD value to greater than the default of 80%. Defining a higher high offload threshold can leave you vulnerable to filling your coupling facility space for the log stream, which means that system logger will reject all write requests until the coupling facility log data can be offloaded to DASD log data sets.

The value specified for HIGHOFFLOAD must be higher than the LOWOFFLOAD value.

,LOWOFFLOAD=*0*
,LOWOFFLOAD=*lowoffload*

Specifies the name (or address in a register) of an 4-byte input field containing the percent value you want to use as the low offload threshold for the coupling facility structure associated with this log stream. The low offload threshold is the target percent where you want offloading to stop, leaving approximately the specified LOWOFFLOAD percentage of log data in the coupling facility structure.

If you specify LOWOFFLOAD=0, which is the default, or omit the LOWOFFLOAD parameter, system logger uses the 0% usage mark as the low offload threshold where offloading stops, leaving 0% of the data in the coupling facility.

The value specified for LOWOFFLOAD must be less than the HIGHOFFLOAD value.

,LIKE=*NO_LIKE*
,LIKE=*like_streamname*

Specifies the name (or address in a register) of a 26-byte input field containing the name of a log stream that has already been defined in the LOGR policy. The characteristics of the already-defined log stream, such as storage class, management class, high level qualifier, and data class will be copied for the log stream you are currently defining. However, the parameters explicitly coded on this request override the characteristics of the log stream specified on the LIKE parameter.

The stream name must be 26 characters, padded on the right with blanks if necessary. The name can be made up of one or more segments separated by periods, up to the maximum length of 26 characters. The following rules apply:

- Each segment may contain up to eight numeric, alphabetic, or national ($, #, or @) characters.
- The first character of each segment must be an alphabetic or national character.
- Each segment must be separated by periods, which you must count as characters.

,MODEL=*NO*
,MODEL=YES

Specifies whether the log stream being defined in the LOGR policy is a model, exclusively for use with the LIKE parameter to set up general characteristics for other log stream definitions.

If you specify MODEL=NO, which is the default, then the log stream being defined is not a model log stream. Systems can connect to and use this log stream. It can also be specified on the LIKE parameter, but is not exclusively for use as a model.

If you specify MODEL=YES, then the log stream being defined is only a model log stream. It can only be specified as a model for other log stream definitions on the LIKE parameter in other IXGINVNT requests.

- Programs **cannot** connect to a log stream name that is defined as a model (MODEL=YES) using an IXGCONN request.
- No log stream data sets are allocated on behalf of a model log stream.
- The attributes of a model log stream are syntax checked at the time of the request, but not verified until a another log stream references the model log stream on the LIKE parameter.

,DIAG=*NO*
,DIAG=YES

Specifies whether or not dumping or additional diagnostics should be provided by Logger for certain conditions. Refer to the DIAG keyword on the IXGCONN, IXGBRWSE and IXGDELET macro services.

If you specify DIAG=NO, which is the default, then no special Logger diagnostic activity is requested for this logstream regardless of the DIAG specifications on the IXGCONN, IXGDELET and IXGBRWSE requests.

If you specify DIAG=YES, then special Logger diagnostic activity is allowed for this logstream and can be obtained when the appropriate specifications are provided on the IXGCONN, IXGDELET or IXGBRWSE requests.

,PLISTVER=*IMPLIED_VERSION*
,PLISTVER=MAX
,PLISTVER=0
,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.
- **0**, which supports all parameters except those specifically referenced in higher versions.
- **1**, which supports both the following parameters and parameters from version 0:
  - DESCRIPTION
  - RMNAME
  - RETPD
  - AUTODELETE
- **2**, which supports both the following parameters and parameters from version 0 and 1:
  - DASDONLY
  - LOGGERDUPLEX

- **3**, which supports the following parameter and parameters from version 0, 1, and 2:
  - EHLQ

To code: Specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, or 3

,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.

REQUEST=DEFINE TYPE=STRUCTURE Option of IXGINVNT

The IXGINVNT macro with the DEFINE TYPE=STRUCTURE parameters defines a coupling facility structure entry in the LOGR policy for a coupling facility log stream.

Syntax for REQUEST=DEFINE TYPE=STRUCTURE

The standard form of the IXGINVNT REQUEST=DEFINE TYPE=STRUCTURE macro is written as follows:

---

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

REQUEST=DEFINE	
,TYPE=STRUCTURE	
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,STRUCTNAME=*structname*	*structname*: RS-type address or register (2) - (12). **Default**: NO_STRUCTNAME
,LOGSNUM=*logsnum*	*logsnum*: RS-type address or register (2) - (12).
,MAXBUFSIZE=*maxbufsize*	*maxbufsize*: RS-type address or register (2) - (12). **Default**: 65532
,AVGBUFSIZE=*avgbufsize*	*avgbufsize*: RS-type address or register (2) - (12).

	Default: 1/2 of *maxbufsize*
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=0 ,PLISTVER=1 ,PLISTVER=2 ,PLISTVER=3	**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=DEFINE,TYPE=STRUCTURE

The parameters are explained as follows:

REQUEST=DEFINE
Requests that an entry for a log stream or coupling facility structure be defined in the LOGR policy.

,TYPE=STRUCTURE
Indicates that the entry to be defined in the LOGR policy is a coupling facility entry being defined for a coupling facility 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.

,STRUCTNAME=*structname*
When specified with TYPE=STRUCTURE, specifies the name (or address in a register) of a 16-byte input field that contains the name of the coupling facility structure you are defining to the LOGR policy.

STRUCTNAME is required for TYPE=STRUCTURE.

The following rules apply for the structname:

- it may contain numeric, alphabetic, or national ($, #, or @) characters, or an underscore(_), padded on the right with blanks if necessary.
- The first character must be an alphabetic character.

,LOGSNUM=*logsnum*

Specifies the name (or address in a register) of a 4-byte input field that contains the number of log streams that can be allocated to the coupling facility structure being defined in the LOGR policy. *logsnum* must be a value between 1 and 512.

IBM recommends that you keep the value for LOGSNUM as small as possible, particularly if your coupling facility structure is small. The more log streams that map to a coupling facility, the less coupling facility space for each log stream and the more chance you stand of running out of space for log streams. See *z/OS MVS Programming: Assembler Services Guide* for more information.

LOGSNUM is required for TYPE=STRUCTURE.

,MAXBUFSIZE=*maxbufsize*

Specifies the name (or address in a register) of a 4-byte input field that contains the size, in bytes, of the largest log block that can be written to log streams allocated to the coupling facility specified in this request.

The value for MAXBUFSIZE must be between 1 and 65,532 bytes. The default is 65,532 bytes.

,AVGBUFSIZE=*avgbufsize*

Specifies the name (or address in a register) of a 4-byte input field of the average size, in bytes, of log blocks written to all the log streams using this coupling facility structure.

System logger uses the average buffer size to control the entry-to-element ratio for this coupling facility structure.

When the active primary LOGR couple data set is at an OS/390 Release 3 level or higher, system logger uses the AVGBUFSIZE specified simply to make an intial determination of the entry-to-element ratio for the structure. After that, system logger monitors structure usage and dynamically manages the entry-to-element ratio accordingly. System logger uses the last entry-to-element ratio in effect for a structure for subseqent structure reallocation requests.

When the active primary LOGR couple data set is at a pre-OS/390 Release 3 level, system logger uses the AVGBUFSIZE specified to calculate an entry-to-element ration that lasts for the life of this coupling facility structure. You cannot update the average buffer size for a structure without first deleting the structure definition (and all the log stream definitions associated with the structure) and then redefining the structure with a new average buffer size.

avgbufsize must be between 1 and the value for MAXBUFSIZE. The default value is 1/2 of the MAXBUFSIZE value.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
,PLISTVER=2
,PLISTVER=3

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.

To code: Specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, or 3

,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.

REQUEST=UPDATE Option of IXGINVNT

The IXGINVNT macro with the UPDATE parameter allows a program to update a log stream entry in the LOGR policy for a coupling facility or DASD-only log stream. Except for the RETPD and AUTODELETE parameters, note that you cannot update a log stream while there are active connections to it.

Syntax for REQUEST=UPDATE

The standard form of the IXGINVNT REQUEST=UPDATE macro is written as follows:

---

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

REQUEST=UPDATE

,TYPE=LOGSTREAM

,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,STREAMNAME=*streamname*	*streamname*: RS-type address or register (2) - (12).
,STRUCTNAME=*structname*	*structname*: RS-type address or register (2) - (12).
,RMNAME=*rmname*	*rmname*: RS-type address or register (2) - (12).
,DESCRIPTION=*description*	*description*: RS-type address or register (2) - (12).
,MAXBUFSIZE=*maxbufsize*	*maxbufsize*: RS-type address or register (2) - (12).
,LOGGERDUPLEX=UNCOND ,LOGGERDUPLEX= COND	**Default**: LOGGERDUPLEX=UNCOND
,STG_DUPLEX=NO ,STG_DUPLEX=YES	
,COND ,UNCOND	
,STG_MGMTCLAS=*stg_mgmtclas*	*stg_mgmtclas*: RS-type address or register (2) - (12).
,STG_DATACLAS=*stg_dataclas*	*stg_dataclas*: RS-type address or register (2) - (12).
,STG_STORCLAS=*stg_storclas*	*stg_storclas*: RS-type address or register (2) - (12).
,STG_SIZE=*stg_size*	*stg_size*: RS-type address or register (2) - (12).
,LS_MGMTCLAS=*ls_mgmtclas*	*ls_mgmtclas*: RS-type address or register (2) - (12).
,LS_DATACLAS=*ls_dataclas*	*ls_dataclas*: RS-type address or register (2) - (12).
,LS_STORCLAS=*ls_storclas*	*ls_storclas*: RS-type address or register (2) - (12).
,LS_SIZE=*ls_size*	*ls_size*: RS-type address or register (2) - (12).
,RETPD=*retpd*	*retpd*: RS-type address or register (2) - (12). **Default**: NO_RETPD
,AUTODELETE=NO ,AUTODELETE=YES	**Default**: AUTODELETE=NO
,LOWOFFLOAD=*lowoffload*	*lowoffload*: RS-type address or register (2) - (12).
,HIGHOFFLOAD=*highoffload*	*highoffload*: RS-type address or register (2) - (12).
,OFFLOADRECALL=NO ,OFFLOADRECALL=YES	**Default**: OFFLOADRECALL=NO_OFFLOADRECALL
,DIAG=NO_DIAG	**Default**: DIAG=NO_DIAG

,DIAG=NO ,DIAG=YES	
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=0 ,PLISTVER=1 ,PLISTVER=2 ,PLISTVER=3	**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=UPDATE

The parameters are explained as follows:

REQUEST=UPDATE
Requests that an entry for a log stream be updated in the LOGR policy.

,TYPE=LOGSTREAM
Requests that the entry to be updated in the LOGR policy is a log stream entry.

,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.

,STREAMNAME=*streamname*
Specifies the name (or address in a register) of the 26-byte input field containing the name of the log stream that you want to define in the LOGR policy.

The stream name must be 26 characters, padded on the right with blanks if necessary. The name can be made up of one or more segments separated by periods, up to the maximum length of 26 characters. The following rules apply:

- Each segment may contain up to eight numeric, alphabetic, or national ($, #, or @) characters.
- The first character of each segment must be an alphabetic or national character.
- Each segment must be separated by periods, which you must count as characters.

STREAMNAME is required with the TYPE=LOGSTREAM parameter.

,STRUCTNAME=*structname*

With REQUEST=UPDATE, specifies the name (or address in a register) of a 16-byte input field containing the name of the coupling facility list structure where all of this log stream's log blocks will be written before being offloaded to DASD. This keyword is allowed when there are no connections (failed or active) to the log stream in the sysplex; otherwise the UPDATE request will be rejected with return code 8, reason code X'0810'.

This keyword can be specified when the existing log stream to be modified is a DASD only log stream. With specification of this keyword, the DASD only log stream will be upgraded to use a coupling facility structure and become a structure-based log stream.

When the active primary LOGR couple data set in the sysplex is formatted at a z/OS R2 level or higher, this keyword can also be specified for a log stream that is currently structure-based in order to upgrade the log stream to a different coupling facility structure. If the LOGR couple data set is not formatted at the appropriate level, the request will fail with return code 8, reason code X'0839'.

STRUCTNAME must be 16 alphanumeric or national ($,#,or @) characters, or underscore (_), padded on the right with blanks if necessary. Lowercase alphabetic characters will be folded to uppercase. The first character must be alphabetic.

Note that the MAXBUFSIZE value in the structure definition for this structure must be equal to or greater than the MAXBUFSIZE specified for the log stream prior to the update. Otherwise, the UPDATE request will be rejected with a return code 8, reason code X'083C'.

,RMNAME=*rmname*

Specifies the name (or address in a register) of the 8-byte input field containing the name of the resource manager program associated with the log stream. RNAME must be 8 alphanumeric or national ($,#,or @) characters, padded on the right with blanks if necessary.

You must define RMNAME in the LOGR policy before the recovery resource manager can connect to the log stream.

If you specify RMNAME to associate a resource manager with a log stream in the LOGR policy, the resource manager specified must subsequently connect to the log stream. If the resource manager does not connect to that log stream, system logger will not process any IXGDELET requests to delete log data. This is so that the resource manager will not miss any delete requests issued against the log stream.

,DESCRIPTION=*description*

Specifies the name (or address in a register) of the 16 character input field containing user defined data describing the log stream.

DESCRIPTION must be 16 alphanumeric or national ($,#,@) characters, underscore (_) or period (.), padded on the right with blanks if necessary.

,MAXBUFSIZE=*maxbufsize*

Specifies the name (or address in a register) of a fullword input field that contains the size, in bytes, of the largest log block that can be written to this DASD-only log stream.

The value for MAXBUFSIZE must be between 1 and 65,532 bytes and cannot be less than the current MAXBUFSIZE for the DASD-only log stream.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. The change will be immediately reflected in the log stream definition, but will not take effect until the subsequent first connection to the DASD-only log stream in the sysplex.

There is no default for the MAXBUFSIZE parameter on an UPDATE request. If you omit this parameter, there will be no change to the MAXBUFSIZE value for this log stream definition.

,LOGGERDUPLEX=UNCOND
,LOGGERDUPLEX=COND

An optional input parameter that specifies whether Logger continues to provide its own log data duplexing, or, conditionally, not provide its own duplexing based on an alternative duplexing configuration that provides an equivalent or better recoverability of the log data.

The active primary TYPE=LOGR couple data set in the sysplex must be formatted at z/OS Release 2 or higher to specify this keyword. Otherwise, the request fails with a return code 8, reason code X'0839'.

This option is not allowed for DASDONLY log streams. Logger continues to unconditionally provide its own duplexing of DASDONLY log stream data.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is formatted at z/OS Release 2 or higher. If a lower format level LOGR couple data set is being used, then the request will fail with a return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex.

Refer to *z/OS MVS Setting Up a Sysplex* sections ″Logger and CF Duplexing Combinations″ and ″System Logger Recovery″ for additional considerations on using the LOGGERDUPLEX parameter.

LOGGERDUPLEX=UNCOND, which is the default, indicates that Logger should provide its own specific duplexing of the log data regardless of any other duplexing (such as structure -system-managed duplexing rebuild) that may be occurring.

LOGGERDUPLEX=COND indicates that Logger should provide its own specific duplexing of the log data unless the log stream is in an alternative duplexing configuration that provides an equivalent or better recoverability of the log data. For example, Logger does not provide its own duplexing of the log data in the following configuration:

- when the log stream is in a non-volatile CF list structure that is handled by system-managed duplexing rebuild. (duplex-mode),
- there is a failure-independent realtionship between the two structure instances, and
- there is a failure-independent connection between connecting system and composite structure view.

,STG_DUPLEX=NO

,STG_DUPLEX=YES

Specifies whether the log stream data for a coupling facility log stream should be considered for duplexing in DASD staging data sets.

Refer to the LOGGERDUPLEX keyword for additional duplexing options.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower format level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex.

Omitting the STG_DUPLEX parameter will not change how the log data resident in the coupling facility for this logstream is duplexed. There is no default for the STG_DUPLEX parameter when requesting an UPDATE to the logstream definition.

If you specify STG_DUPLEX=NO, log data for a coupling facility log stream is not duplexed in staging data sets, regardless of the failure independence/dependence coupling facility. A coupling facility is considered failure independen when it is non-volatile and resides on a different CPC from the MVS image using it. Otherwise, the coupling facility is failure dependent.

When the update to STG_DUPLEX=NO takes effect, the staging data set related parameters (DUPLEXMODE, STG_SIZE, STG_DATACLAS, STG_MGMTCLAS, and STG_STORCLAS) are cleared in the logstream definition.

If you specify STG_DUPLEX=YES, the log data for a coupling facility log stream will be duplexed in staging data sets if the conditions defined by the DUPLEXMODE parameter are fulfilled.

STG_DUPLEX is only valid for a coupling facility log stream.

There is no default for the STG_DUPLEX parameter on an UPDATE request. If you omit this parameter, there is no change to the staging duplexing status for the log stream definition.

You can use the DUPLEXMODE parameter with STG_DUPLEX and with LOGGERDUPLEX to specify the type of duplexing desired and whether you want conditional or unconditional duplexing by Logger.

,DUPLEXMODE=COND
,DUPLEXMODE=UNCOND

Specifies the conditions under which the coupling facility log data for a coupling facility log stream should be duplexed in DASD staging data sets.

If you specify DUPLEXMODE=COND, the coupling facility log data will be duplexed in staging data sets only if a system's connection to the coupling facility log stream contains a single point of failure and is therefore vulnerable to permanent log data loss:

- A connection to a log stream contains a single point of failure if the coupling facility is volatile and/or resides on the same CPC as the MVS system connecting to it. The coupling facility log data for the system connection containing the single point of failure will be duplexed to staging data sets..
- A connection to a log stream is failure-independent when the coupling facility for the log stream is non-volatile and resides on a different central processor complex (CPC) than the MVS system connecting to it. The coupling facility log data for that system connection will not be duplexed to staging data sets..

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 level. If a lower format level LOGR couple data set is being used, then the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex.

If you specify DUPLEXMODE=UNCOND, the log data for the coupling facility log stream will be duplexed in staging data sets, unconditionally, even if the connection is failure independent.

There is no default for the DUPLEXMODE parameter on an UPDATE request. If you omit this parameter, there will be no change to the duplexing mode for the coupling facility log stream definition.

You can use the DUPLEXMODE parameter with STG_DUPLEX and with LOGGERDUPLEX to specify the type of duplexing desired and whether you want conditional or unconditional duplexing by Logger.

See the *z/OS MVS Setting Up a Sysplex* section on Duplexing Log Data for complete information on using staging data sets to duplex coupling facility log data.

DUPLEXMODE is valid only when STG_DUPLEX=YES has been specified for a coupling facility log stream.

,STG_DATACLAS=*stg_dataclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS data class that will be used for allocation of the DASD staging data set for this log stream.

The data class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

An SMS value specified on the STG_DATACLAS parameter, including NO_STG_DATACLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

There is no default for the STG_DATACLAS parameter on an UPDATE request. If you omit this parameter, there will be no change to the data class for staging data sets for this log stream definition.

STG_DATACLAS is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,STG_MGMTCLAS=*stg_mgmtclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS management class that will be used for allocation of the DASD staging data set for this log stream.

The management class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

An SMS value specified on the STG_MGMTCLAS parameter, including NO_STG_MGMTCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

There is no default for the STG_MGMTCLAS parameter on an UPDATE request. If you omit this parameter, there will be no change to the management class for staging data sets for this log stream definition.

STG_MGMTCLAS is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,STG_STORCLAS=*stg_storclas*
Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS storage class that will be used for allocation of the DASD staging data set for this log stream.

The storage class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

An SMS value specified on the STG_STORCLAS parameter, including NO_STG_STORCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

There is no default for the STG_STORCLAS parameter on an UPDATE request. If you omit this parameter, there will be no change to the storage class for staging data sets in this log stream definition.

STG_STORCLAS is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,STG_SIZE=*stg_size*
Specifies the name (or address in a register) of a 4-byte input field containing the size, in 4K blocks, of the DASD staging data set for this log stream.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

If you omit this parameter, there will be no change to the DASD staging data size in this log stream definition. Note that if both the STG_DATACLAS and STG_SIZE are specified, the value for STG_SIZE overrides the space allocation attributes for the data class specified on the STG_DATACLAS value.

STG_SIZE is only valid with STG_DUPLEX=YES or DASDONLY=YES.

,LS_DATACLAS=*ls_dataclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS data class that will be used for allocation of the DASD log data set for this log stream.

The data class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect when the next log stream offload data set is allocated (data set switch event) or on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

An SMS value specified on the LS_DATACLAS parameter, including NO_LS_DATACLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

There is no default for the LS_DATACLAS parameter on an UPDATE request. If you omit this parameter, there will be no change to the data class for the log stream data sets for this log stream definition.

,LS_MGMTCLAS=*ls_mgmtclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS management class that will be used for allocation of the DASD log data set for this log stream.

The management class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect when the next log stream offload data set is allocated (data set switch event) or on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

An SMS value specified on the LS_MGMTCLAS parameter, including NO_LS_MGMTCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

There is no default for the LS_MGMTCLAS parameter on an UPDATE request. If you omit this parameter, there will be no change to the management class for the log stream data sets for this log stream definition.

,LS_STORCLAS=*ls_storclas*

Specifies the name (or address in a register) of an 8-byte input field containing the name of the SMS storage class that will be used for allocation of the DASD log data set for this log stream.

The storage class must be 8 alphanumeric or national ($,#, or @) characters, padded on the right with blanks if necessary. The first character must be an alphabetic or national character.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect when the next log stream offload data set is allocated (data set switch event) or on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

An SMS value specified on the LS_STORCLAS parameter, including NO_LS_STORCLAS, **always** overrides one specified on a model log stream used on the LIKE parameter.

There is no default for the LS_STORCLAS parameter on an UPDATE request. If you omit this parameter, there will be no change to the storage class for the log stream data sets for this log stream definition.

,LS_SIZE=*ls_size*

Specifies the name (or address in a register) of a fullword input field containing the size, in 4K blocks, of the DASD log data set for the log stream being defined.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower fomat level LOGR couple data set is being used, the request will fail with return code 8, reason code X'0810'. The change will be immediately reflected in the log stream definition. It will take effect when the next log stream offload data set is allocated (data set switch event) or on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

If you omit this parameter, there will be no change to the DASD log data set size for the log stream being updated.

Note that a value specified on the LS_SIZE parameter overrides the space allocation attributes for the data class specified on the LS_DATACLAS parameter.

,AUTODELETE=NO
,AUTODELETE=YES

Specifies when system logger physically deletes log data from the log stream.

This keyword can be updated regardless of whether the log stream is actively connected or not. The change will be immediately reflected in the log stream definition. It will take effect upon the next data set switch event or on the subsequent first connection to the log stream in the sysplex. In order to specify this keyword, the LOGR couple data set must be formatted at an OS/390 R3 level or higher; otherwise, the request will fail with return code 8, reason code X'0839'.

If you specify AUTODELETE=NO, which is the default, system logger physically deletes an entire log data set only when **both** of the following are true:

- Data is marked for deletion by a system logger application using the IXGDELET service.
- The retention period for all the data in the log data set expires.

If you specify AUTODELETE=YES, system logger automatically physically deletes log data whenever data is either marked for deletion (using the IXGDELET service or an archiving procedure) or the retention period for all the log data in a data set has expired.

Be careful when using AUTODELETE=YES if the system logger application manages log data deletion using the IXGDELET service. With AUTODELETE=YES, system logger may delete data that the application expects to be accessible.

The LOGR couple data set must be formatted at the OS/390 Release 3 level or above to use this keyword.

RETPD=0
RETPD=*retpd*

Specifies the name (or address in a register) of a fullword input field containing the number of days of the retention period for log data in the log stream. The retention period begins when data is written to the log stream. Once the retention period for an entire log data set has expired, the data set is eligible for physical deletion. The point at which system logger physically deletes the data depends on what you have specified on the AUTODELETE parameter. System logger will not process a retention period or delete data on behalf of log streams that are not connected to or being written to by an application.

This keyword can be updated regardless of whether the log stream is actively connected or not. The change will be immediately reflected in the log stream definition. It will take effect upon the next data set switch event or on the subsequent first connection to the log stream in the sysplex. In order to specify this keyword, the LOGR couple data set must be formatted at an OS/390 R3 level or higher; otherwise, the request will fail with return code 8, reason code X'0839'.

The value specified for RETPD must be between 0 and 65,536.

,HIGHOFFLOAD=*highoffload*

Specifies the name (or address in a register) of a fullword input field containing the percent value you want to use as the high offload threshold for the coupling facility structure associated with this log stream. When the coupling facility is filled to the high offload threshold point or beyond, system logger begins offloading data from the coupling facility to the DASD log stream data sets.

IBM recommends that you are careful in considering to define your HIGHOFFLOAD value to greater than 80%. Defining a higher high offload threshold can leave you vulnerable to filling your coupling facility space for the log stream, which means that system logger will reject all write requests until the coupling facility log data can be offloaded to DASD log data sets.

The value specified for HIGHOFFLOAD must be higher than the LOWOFFLOAD value.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower format level LOGR couple data set is being used, then the request will fail with return code 8, reason code X'0810'. The change will immediately be reflected in the log stream definition. It will take effect on the subsequent first

connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild. For a DASD-only log stream, the change will take effect upon the next offload data set switch event.

There is no default for the HIGHOFFLOAD parameter on an UPDATE request. If you omit this parameter, there will be no change to the high offload value for this log stream definition.

,LOWOFFLOAD=*lowoffload*

Specifies the name (or address in a register) of a fullword input field containing the percent value you want to use as the low offload threshold for the coupling facility structure associated with this log stream. The low offload threshold is the target percent where you want offloading to stop, leaving approximately the specified LOWOFFLOAD percentage of log data in the coupling facility structure.

The value specified for LOWOFFLOAD must be less than the HIGHOFFLOAD value.

This keyword can be updated even when the log stream is actively connected when the LOGR couple data set is at least at the z/OS Release 2 format level. If a lower format level LOGR couple data set is being used, then the request will fail with return code 8, reason code X'0810'. The change will immediately be reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild. For a DASD-only log stream, the change will take effect upon the next offload data set switch event.

There is no default for the LOWOFFLOAD parameter on an UPDATE request. If you omit this parameter, there will be no change to the low offload value for this log stream definition.

,OFFLOADRECALL=NO_OFFLOADRECALL
,OFFLOADRECALL=YES
,OFFLOADRECALL=NO

Specifies whether or not offload processing is to skip recalling the current offload data set. In order to use UPDATE with OFFLOADRECALL, the LOGR couple data set must be formatted at an OS/390 Release 3 or higher level.

This keyword can be updated even when the log stream is actively connected. The change will immediately be reflected in the log stream definition. It will take effect on the subsequent first connection to the log stream in the sysplex. For a structure-based log stream, the change will also take effect during the next structure rebuild.

Specifying OFFLOADRECALL=NO_OFFLOADRECALL indicates that the OFFLOADRECALL attribute of the log stream should not be updated.

Specifying OFFLOADRECALL=YES indicates that offload processing should recall the current offload data set.

Specifying OFFLOADRECALL=NO indicates that offload processing should not recall the current offload data set and allocate a new one. Note that this option may cause any or all of the current offload data set to be wasted space on DASD once it is recalled. Care should be taken when using this option to size the data sets appropriately.

,DIAG=NO_DIAG
,DIAG=NO

,DIAG=YES

Specifies whether or not dumping or additional diagnostics should be provided by Logger for certain conditions. Refer to the DIAG keyword on the IXGCONN, IXGBRWSE and IXGDELET macro services.

If you specify DIAG=NO, which is the default, then no special Logger diagnostic activity is requested for this logstream regardless of the DIAG specifications on the IXGCONN, IXGDELET and IXGBRWSE requests.

If you specify DIAG=YES, then special Logger diagnostic activity is allowed for this logstream and can be obtained when the appropriate specifications are provided on the IXGCONN, IXGDELET or IXGBRWSE requests.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
,PLISTVER=2
,PLISTVER=3

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:
  - DESCRIPTION
  - RMNAME
  - RETPD
  - AUTODELETE
- **2**, which supports both the following parameters and parameters from version 0 and 1:
  - DASDONLY
  - LOGGERDUPLEX
- **3**, which supports the following parameter and parameters from version 0, 1, and 2:
  - EHLQ

To code: Specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, or 3

,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.

REQUEST=DELETE Option of IXGINVNT

The IXGINVNT macro with the DELETE parameter allows a program to delete a log stream entry or coupling facility structure entry in the LOGR policy.

Syntax for REQUEST=DELETE

The IXGINVNT REQUEST=DELETE macro is written as follows:

---

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

REQUEST=DELETE	
,TYPE=LOGSTREAM ,TYPE=STRUCTURE	
,ANSAREA=*ansarea*	*ansarea*: RS-type address or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or register (2) - (12).
,STREAMNAME=*streamname*	*streamname*: RS-type address or register (2) - (12).
,STRUCTNAME=*structname*	*structname*: RS-type address or register (2) - (12). **Default**: NO_STRUCTNAME
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=0 ,PLISTVER=1 ,PLISTVER=2 ,PLISTVER=3	**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	**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=DELETE

The parameters are explained as follows:

REQUEST=DELETE
Requests that an entry for a log stream or coupling facility structure be deleted from the LOGR policy.

,TYPE=LOGSTREAM
Requests that the entry to be deleted from the LOGR policy is a log stream entry.

If you specify TYPE=LOGSTREAM, you must also specify STREAMNAME, ANSAREA, and ANSLEN.

,TYPE=STRUCTURE
Requests that the entry to be deleted from the LOGR policy is a coupling facility entry.

If you specify TYPE=STRUCTURE, you must also specify STRUCTNAME, ANSAREA, and ANSLEN.

,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.

,STREAMNAME=*streamname*
Specifies the 26-byte field (or address in a register) of the log stream that you want to delete from the LOGR policy.

The stream name must be 26 characters, padded on the right with blanks, if necessary. The name can be made up of one or more segments, up to the maximum length of 26 characters. The following rules apply:

- Each segment may contain 1-8 numeric, alphabetic, or national ($, #, or @) characters.
- The first character of each segment must be an alphabetic or national character.
- Each segment must be separated by periods, which count as characters.

STREAMNAME is required for TYPE=LOGSTREAM.

,STRUCTNAME=*structname*
Specify TYPE=STRUCTURE to specify the name (or address in a register) of a 16-byte input field that contains the name of the coupling facility structure you are deleting from the LOGR policy.

STRUCTNAME is required for TYPE=STRUCTURE.

The following rules apply for the structname:

- It may contain numeric, alphabetic, or national ($, #, or @) characters, or an underscore(_), padded on the right with blanks if necessary
- The first character must be an alphabetic character.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
,PLISTVER=1
,PLISTVER=2
,PLISTVER=3
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.

To code: Specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value of 0, 1, 2, or 3

,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 IXGINVNT 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 IXGCONN 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 20. Return and Reason Codes for the IXGINVNT Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**Equate Symbol**: IxgRsnCodeOk **Explanation**: Request processed successfully.
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	xxxx0805	**Equate Symbol**: IxgRsnCodeAllocError **Explanation**: Environment error. The system encountered a severe dynamic allocation error while processing data sets related to the log stream. To obtain further information about the error check the SYSLOG or OPERLOG for message IXG251I or any other data set related message. ANSAA_DIAG1 of the answer area contains either the dynamic allocation return code or an internal Logger return code. ANSAA_DIAG2 of the answer area contains either the dynamic allocation error code or the DFSMS reason code. **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	xxxx0808	**Equate Symbol**: IxgRsnCodeIOError **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.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx080A	**Equate Symbol**: IxgRsnCodeRequestLocked **Explanation**: Program error. The program issuing the request is holding a lock. **Action**: Ensure that the program issuing the request is not holding a lock.
08	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	xxxx080D	**Equate Symbol**: IxgRsnCodeNoSAFAuth **Explanation**: Environment error. The user does not have correct SAF authorization for the request. The caller is not authorized for one of the following: • The log stream being updated or defined. • The log stream named on the LIKE parameter. • The structure specified. **Action**: Define SAF authorization for any log streams and structures specified.
08	xxxx080E	**Equate Symbol**: IxgRsnCodeStreamDefined **Explanation**: Program error. The log stream name specified already been defined in the LOGR inventory couple data set. **Action**: Do one of the following: • Use the existing definition for the log stream. • Change the name of the log stream being defined. • Delete the existing log stream definition from the inventory and then reissue the IXGINVNT request to redefine it.
08	xxxx0810	**Equate Symbol**: IxgRsnCodeStreamInuse **Explanation**: Environment error. You cannot alter or delete a log stream while an application is connected to it. Some attributes can be updated while there are connections provided the appropriate LOGR couple data set and release levels are in effect. **Action**: Reissue the request when there are no active connections to the log stream or move to the appropriate release and LOGR couple data set format level.
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 20. Return and Reason Codes for the IXGINVNT 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**: Reissue 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	xxxx0819	**Equate Symbol**: IxgRsnCodeSRBMode **Explanation**: Program error. The calling program is in SRB mode, but task mode is the required dispatchable unit mode for this system logger service. **Action**: Make sure the calling program is in task mode.
08	xxxx081A	**Equate Symbol**: IxgRsnCodeMaxStreamConn **Explanation**: Environment error. This system has reached the limit for the maximum number of log streams that can be concurrently active. System logger allows 4096 concurrently active log streams per system. **Action**: Your work load may need to be planned to either consolidate log streams or balance system activity such that fewer log streams are needed during this time frame.
08	xxxx081B	**Equate Symbol**: IxgRsnCodePrimaryNotHome **Explanation**: Program error. The primary address space does not equal the home address space. **Action**: Make sure that the primary address space equals the home address space when issuing this system logger service.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
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).
08	xxxx0821	**Equate Symbol**: IxgRsnCodeDspCreateFailed **Explanation**: System error. A data space create failed during inventory report processing. **Action**: See ANSAA_DIAG1 for the DSPSERV return code and ANSAA_DIAG2 for the DSPSERV reason code.
08	xxxx0822	**Equate Symbol**: IxgRsnCodeBadHlq **Explanation**: Program error. The high level qualifier specified on the HLQ parameter was incorrect. **Action**: Specify a valid high level qualifier and reissue the request.
08	xxxx0823	**Equate Symbol**: IxgRsnCodeNoInvrecSpace **Explanation**: Environment error. The LOGR couple data set cannot be updated because the maximum number of entries for the specified type has already been reached. **Action**: • Format a new LOGR couple data set using the IXCL1DSU utility. In the new LOGR couple data set either delete unused entries or increase the allowed number of entries on the LSR parameter (for log stream entries) or the LSTRR parameter (for coupling facility structure entries). • PSWITCH the current alternate LOGR couple data set to primary. • Add the new LOGR couple data set as alternate. • PSWITCH the new LOGR couple data set from alternate to primary.
08	xxxx0824	**Equate Symbol**: IxgRsnCodeMaxStreamStr **Explanation**: Program error. A program issued IXGINVNT to associate a structure with a log stream, but the maximum number of log streams allowed (as defined on the LOGSNUM parameter) has been reached for the specified structure. **Action**: Either specify a structure that has not reached its LOGSNUM limit, or specify a larger LOGSNUM value on the definition for the structure.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0825	**Equate Symbol**: IxgRsnCodeStrDefined **Explanation**: Program error. The structure specified on the IXGINVNT request is already defined in the LOGR inventory couple data set. **Action**: Either use the existing structure definition, change the name of the structure being defined or delete the existing structure and redefine it.
08	xxxx0826	**Equate Symbol**: IxgRsnCodeBadLogsnum **Explanation**: Program error. The LOGSNUM value specified for a structure definition was not within the valid range between 1 and 512. **Action**: Change the LOGSNUM value to be within the valid range.
08	xxxx0827	**Equate Symbol**: IxgRsnCodeNoStrRecord **Explanation**: Program error. The coupling facility structure specified in the definition for a log stream is not defined in the LOGR inventory couple data set. **Action**: Either define the coupling facility structure before referencing it in a log stream definition, or specify an existing structure definition.
08	xxxx0828	**Equate Symbol**: IxgRsnCodeStrRecordInuse **Explanation**: Program error. The request to delete a structure definition from the LOGR inventory couple data set cannot be completed because several log stream definitions reference it. You cannot delete a structure definition until all the log streams associated with it have been deleted first. **Action**: Delete all the log streams associated with the structure you wish to delete, then reissue the request.
08	xxxx0829	**Equate Symbol**: IxgRsnCodeBadStgStorClas **Explanation**: Program error. The name specified on the STG_STORCLAS parameter is incorrect. **Action**: Change the staging data set storage class specified to meet the STG_STORCLAS syntax requirements.
08	xxxx082A	**Equate Symbol**: IxgRsnCodeBadLSStorClas **Explanation**: The name specified on the LS_STORCLAS parameter is incorrect. **Action**: Change the log stream data set storage class specified to meet the LS_STORCLAS syntax requirements.
08	xxxx082B	**Equate Symbol**: IxgRsnCodeBadStreamLike **Explanation**: Program error. The log stream name specified on the LIKE parameter was not valid. **Action**: Reissue the request with a valid log stream name on the LIKE parameter.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx082C	**Equate Symbol**: IxgRsnCodeBadStructName **Explanation**: Program error. The coupling facility structure name specified on the STRUCTNAME parameter is not valid. **Action**: Reissue the request with a valid structure name on the STRUCTNAME parameter.
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	xxxx082F	**Equate Symbol**: IxgRsnCodeBadStgDataClas **Explanation**: Program error. The name specified on the LS_DATACLAS parameter is not valid. **Action**: Change the data class specified to meet the LS_DATACLAS syntax requirements.
08	xxxx0830	**Equate Symbol**: IxgRsnCodeBadLSDataClas **Explanation**: Program error. The name specified on the STG_DATACLAS parameter is not valid. **Action**: Change the data class specified to meet the STG_DATACLAS syntax requirements.
08	xxxx0831	**Equate Symbol**: IxgRsnCodeBadStreamName **Explanation**: Program error. The log stream name specified on the STREAMNAME parameter is not valid. **Action**: Reissue the request with a valid log stream name on the STREAMNAME parameter.
08	xxxx0832	**Equate Symbol**: IxgRsnCodeBadStgMgmtClas **Explanation**: Program error. The name specified on the STG_MGMTCLAS parameter is not valid. **Action**: Change the staging data set management class specified to meet the STG_MGMTCLAS syntax requirements.
08	xxxx0833	**Equate Symbol**: IxgRsnCodeBadLSMgmtClas **Explanation**: Program error. The name specified on the LS_MGMTCLAS parameter is not valid. **Action**: Change the log stream data set management class specified to meet the LS_MGMTCLAS syntax requirements.
08	xxxx0834	**Equate Symbol**: IxgRsnCodeInvalidLSSize **Explanation**: Program error. A non-zero LS_SIZE is specified, but is not in the range valid for a VSAM linear data set. **Action**: Either change the LS_SIZE or omit it from the DEFINE request to accept the default value.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0835	**Equate Symbol**: IxgRsnCodeInvalidStgSize **Explanation**: Program error. A non-zero STG_SIZE is specified, but is not in the range valid for a VSAM linear data set. **Action**: Either change the STG_SIZE or omit it from the DEFINE request to accept the default value.
08	xxxx0838	**Equate Symbol**: IxgRsnCodeUnDefSmsClas **Explanation**: Program error. At least one of the names specified for DATACLAS, MGMTCLAS, or STORCLAS is not defined to SMS. **Action**: Specify names that are defined to the active SMS configuration.
08	xxxx0839	**Equate Symbol**: IxgRsnCodeBadCdsLevel **Explanation**: The active primary LOGR couple data set is not formatted at the level required for the request. See the explanation of the parameters for the level each requires. **Action**: Do one of the following: • Bring a new new active primary LOGR couple data set at the required level into the sysplex and then retry the request. • Remove the keywords requiring an new level of the LOGR couple data set and retry the request.
08	xxxx083C	**Equate Symbol**: IxgRsnCodeBadMaxBufSize **Explanation**: Program error. For a DEFINE or UPDATE request, the value specified for MAXBUFSIZE was incorrect. It must be a value between 1 and 65,532. For an UPDATE request, one of the following is causing the error: • The value specified is less than the MAXBUFSIZE value currently associated with a DASD-only log stream, or • The current DASD-only MAXBUFSIZE value is greater than the MAXBUFSIZE value associated with the STRUCTNAME specified on the update request, or • The current structure MAXBUFSIZE value is greater than the MAXBUFSIZE value associated with the STRUCTNAME specified on the UPDATE request. **Action**: Do one of the following, depending on the request: **For a DEFINE request**, specify a valid value for MAXBUFSIZE and reissue the request. **For an UPDATE request,** do one of the following: • Specify a value within the valid range for MAXBUFSIZE that is greater than or equal to the current DASD-only MAXBUFSIZE value. • Ensure that the structure specified on the STRUCTNAME parameter has a maximum buffer size that is greater than or equal to the current MAXBUFSIZE value associated with the log stream specified on the UPDATE request.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx083E	**Equate Symbol**: IxgRsnCodeNoAvailSysRec **Explanation**: System error. There were no available system records. **Action**: Contact the IBM support center. Provide the return and reason codes and the contents of the system logger trace.
08	xxxx0840	**Equate Symbol**: IxgRsnCodeBadVersion **Explanation**: Environment error. The parameter list passed to the service routine had an invalid version indicator. **Action**: Ensure the level of MVS executing the request and the macro library used to compile the invoking routine are compatible.
08	xxxx0842	**Equate Symbol**: IxgRsnCodeBadAvgBufSize **Explanation**: Program error. The value specified for AVGBUFSIZE was specified as incorrect. It must be a value between and 65,536 that is less than MAXBUFSIZE. **Action**: Reissue the request with a valid AVGBUFSIZE value.
08	xxxx084E	**Equate Symbol**: IxgRsnCodeStrSpaceTooSmall **Explanation**: Environment error. Structure resources are not available to satisfy the request. All structure resources are allocated as system logger control resources. This condition occurs when the structure resources are consumed by the logstreams connection. **Action**: Increase the size of the structure in the CFRM policy, or use SETXCF ALTER support to dynamically increase the size of the structure.
08	xxxx0843	**Equate Symbol**: IxgRsnCodeXcdsReformat **Explanation**: Program error. A couple data set record is not valid. **Action**: Reformat the system logger couple data set.
08	xxxx0844	**Equate Symbol**: IxgRsnCodeNoStreamLike **Explanation**: Program error. The log stream name specified on the LIKE parameter is not defined in the LOGR couple data set. **Action**: Do one of the following: • Define the log stream you wish to reference in the LOGR inventory couple data set and reissue the request. • Reissue the request, specifying a different log stream that is already defined in the LOGR couple data set.
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 reissue the request.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
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 may 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 least CFLEVEL=1.
08	xxxx0853	**Equate Symbol**: IxgRsnCodeNoCF **Explanation**: The connect request was rejected. System logger could not allocate coupling facility structure space, because no suitable coupling facility was available. **Action**: Check accompanying message IXG206I for a list of the coupling facilities, where space allocation was attempted and the reason why each attempt failed.
08	xxxx0854	**Equate Symbol**: IxgRsnCodeBadLowoffload **Explanation**: Program error. The value specified for LOWOFFLOAD is not valid. **Action**: Change the value to meet the LOWOFFLOAD syntax requirements.
08	xxxx0855	**Equate Symbol**: IxgRsnCodeBadHighoffload **Explanation**: Program error. The value specified for HIGHOFFLOAD is invalid. **Action**: Change the value to meet the HIGHOFFLOAD syntax requirements.
08	xxxx0856	**Equate Symbol**: IxgRsnCodeBadLowHighOffLoad **Explanation**: Program error. The value specified or defaulted to for the low offload value is equal to or higher than the high offload value. The low offload value must be lower than the high offload value. **Action**: Change either the LOWOFFLOAD parameter or the HIGHOFFLOAD parameter so that the low offload value is less than the high offload value.
08	xxxx0857	**Equate Symbol**: IxgRsnCodeDuplexmodeDuplexNo **Explanation**: Program error. DUPLEXMODE was specified, but the log stream was defined with STG_DUPLEX=NO. The DUPLEXMODE parameter is only valid with STG_DUPLEX=YES. **Action**: Either change the log stream definition to specify STG_DUPLEX=YES or else omit DUPLEXMODE from the request.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0858	**Equate Symbol**: IxgRsnCodeStgsizeDuplexNo **Explanation**: Program error. A non-zero STG_SIZE is specified but the log stream is defined with STG_DUPLEX=NO. **Action**: Either change the log stream definition to specify STG_DUPLEX=YES, or else omit the non-zero STG_SIZE from the request.
08	xxxx0859	**Equate Symbol**: IxgRsnCodeDataClasDuplexNo **Explanation**: Program error. A non-blank STG_DATACLAS is specified but the log stream is defined with STG_DUPLEX=NO. **Action**: Either change the log stream definition to specify STG_DUPLEX=YES. or else omit the non-blank STG_DATACLAS from the request.
08	xxxx085A	**Equate Symbol**: IxgRsnCodeMgmtClasDuplexNo **Explanation**: Program error. A non-blank STG_MGMTCLAS is specified but the log stream is defined with STG_DUPLEX=NO. **Action**: Either change the log stream definition to specify STG_DUPLEX=YES, or else omit the non-blank STG_MGMTCLAS from the request.
08	xxxx085B	**Equate Symbol**: IxgRsnCodeStorClasDuplexNo **Explanation**: Program error. A non-blank STG_STORCLAS is specified but the log stream was defined with STG_DUPLEX=NO. **Action**: Either change the log stream definition to specify STG_DUPLEX=YES, or else omit the non-blank STG_STORCLAS from the request.
08	xxxx085E	**Equate Symbol**: IxgRsnCodeNoStructName **Explanation**: Program error. A structure name was not provided for this log stream via the STRUCTNAME parameter or defined for a log stream named on a LIKE parameter. A STRUCTNAME value is required to successfully define a log stream to the LOGR couple data set. **Action**: Provide a value for the STRUCTNAME parameter or define a structure for the log stream referenced on the LIKE parameter.
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 20. Return and Reason Codes for the IXGINVNT 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. 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	xxxx08D4	**Equate Symbol**: IxgRsnCodeBadRMName **Explanation**: Program Error. The name of the resource manager specified on the RMNAME parameter was not valid. **Action**: Correct the RMNAME and retry the request.
08	xxxx08D5	**Equate Symbol**: IxgRsnCodeBadLSDescription **Explanation**: Program Error. The name of the field specified in the DESCRIPTION parameter was not valid. DESCRIPTION must be 16 alphanumeric or national ($,#,@) characters, underscore (_) or period (.), padded on the right with blanks if necessary. **Action**: Correct the DESCRIPTION field name and retry the request.
08	xxxx08D8	**Equate Symbol**: IxgRsnCodeBadRetpd **Explanation**: Program Error. The value specified for RETPD was incorrect. It must be a value >= 0 and <= 65,536. **Action**: Specify a valid value for RETPD and reissue the request.
08	xxxx08E0	**Equate Symbol**: IxgRsnCodeStgDuplexDasdOnly **Explanation**: Program Error. The STG_DUPLEX parameter was specified on an UPDATE request, but the log stream specified was defined as DASDONLY=YES. STG_DUPLEX is only valid with a coupling facility log stream (DASDONLY=NO). **Action**: Either correct the request to specify a coupling facility log stream or eliminate the STG_DUPLEX parameter on this request.
08	xxxx08E1	**Equate Symbol**: IxgRsnCodeDuplexModeDasdOnly **Explanation**: Program Error. The DUPLEXMODE parameter was specified on an UPDATE request, but the log stream specified was defined as DASDONLY=YES. DUPLEXMODE is only valid with a coupling facility log stream (DASDONLY=NO). **Action**: Either correct the request to specify a coupling facility log stream or eliminate the DUPLEXMODE parameter on this request.

Table 20. Return and Reason Codes for the IXGINVNT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx08E3	**Equate Symbol**: IXGRSNCODELOGSTREAMNOTSUPPORTED **Explanation**: Environment error. A request to define or alter a log stream definition in the LOGR policy was rejected on this system because the system release level does not support a DASD-only log stream. stream. The system must be at OS/390 Release 4 or higher to define, update, or delete a DASD-only log stream. **Action**: If you must issue a request against a DASD-only log stream, make sure you issue it from a system that is OS/390 Release 4 or higher. Otherwise, define the log stream as a coupling facility log stream by specifying a structure name in the log stream definition.
08	xxxx08E4	**Equate Symbol**: IXGRSNCODEMAXBUFSIZEDASDONLY **Explanation**: Program error. A value was specified for MAXBUFSIZE on this request, but the log stream was defined as a coupling facility log stream (DASDONLY=NO). MAXBUFSIZE is not a valid parameter on a log stream definition request for a coupling facility log stream. **Action**: Either remove the MAXBUFSIZE parameter from this request or specify DASDONLY=YES with MAXBUFSIZE.
08	xxxx08E5	**Equate Symbol**: IxgRsnCodeLoggerDuplexDasdOnly **Explanation**: Program error. The LOGGERDUPLEX parameter was specified on a define or an update request, but the log stream was defined with DASDONLY=YES. The LOGGERDUPLEX parameter is only valid for a coupling facility structure based log stream. **Action**: Do not specify the log stream LOGGERDUPLEX parameter on a define or an update request for a DASD only log stream.
08	xxxx08E6	**Equate Symbol**: IxgRsnCodeBadEhlq **Explanation**: Program error. The extended high level qualifier for the log stream data sets specified on the EHLQ parameter was incorrect. This could be from a syntax error or by specifying EHLQ and HLQ on the same request. **Action**: Specify a valid extended high level qualifier (EHLQ) or high level qualifier (HLQ) and reissue the request.
08	xxxx08E7	**Equate Symbol**: IxgRsnCodeEhlqTooLong **Explanation**: Program error. The combined length of the extended high level qualifier (EHLQ value) and the log stream name (with a period delimiter) exceeds 35 characters. The combined length of the EHLQ value, the log stream name, and the logger suffix (with period delimiters) cannot exceed 44 characters. **Action**: Specify a valid extended high-level qualifier (EHLQ) or high-level qualifier (HLQ) and reissue the request.

Table 20. Return and Reason Codes for the IXGINVNT 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.

Example 1

Issue IXGINVNT REQUEST=DEFINE to define a coupling facility structure associated with one or more log streams.

```
          IXGINVNT REQUEST=DEFINE,                                    X
               TYPE=STRUCTURE,                                        X
               STRUCTNAME=STRUCT,                                     X
               LOGSNUM=LOGNUM,                                        X
               AVGBUFSIZE=AVGBUF,                                     X
               MAXBUFSIZE=MAXBUF,                                     X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
STRUCT   DC    CL16'LIST01'           structure name
LOGNUM   DC    F'10'                  num allocated logstreams allowed
AVGBUF   DC    F'256'                 average buffer size
MAXBUF   DC    F'4096'                maximum buffer size
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
RETCODE  DS    F                      return code from logger
RSNCODE  DS    F                      reason code from logger
DATAREA  DSECT
         IXGANSAA LIST=YES               answer area
```

Example 2

Issue IXGINVNT REQUEST=DEFINE to define a log stream that writes to both the coupling facility and DASD log data sets as a model and issue IXGINVNT REQUEST=DEFINE a second time to define another log stream modeled on the first using the LIKE parameter.

```
          IXGINVNT REQUEST=DEFINE,                                    X
               TYPE=LOGSTREAM,                                        X
               STREAMNAME=STRNAME,                                    X
               STRUCTNAME=STRUCT,                                     X
               DATACLAS=DATACLAS,                                     X
               MGMTCLAS=MGMTCLAS,                                     X
               STORCLAS=STORCLAS,                                     X
               HLQ=HLQ,                                               X
               MODEL=YES,                                             X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
```

```
          IXGINVNT REQUEST=DEFINE,                                    X
               TYPE=LOGSTREAM,                                        X
               STREAMNAME=STRNAME1,                                   X
               LIKE=STRNAME,                                          X
               STRUCTNAME=STRUCT,                                     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
STRNAME  DC    CL26'LOG.STREAM.NAME'  stream name for model
STRNAME1 DC    CL26'LOG.STREAM1.NAME' stream name for like
STRUCT   DC    CL16'LIST01'           associated structure name
DATACLAS DC    CL8'VSAMLS'            data class name
MGMTCLAS DC    CL8'INTERIM'           management class name
STORCLAS DC    CL8'STANDARD'          storage class name
HLQ      DC    CL8'USERNAME'          high level qualifier
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 3

Issue IXGINVNT REQUEST=UPDATE to update a log stream definition.

```
          IXGINVNT REQUEST=UPDATE,                                    X
               TYPE=LOGSTREAM,                                        X
               STREAMNAME=STRNAME,                                    X
               DATACLAS=DATACLAS,                                     X
               MGMTCLAS=MGMTCLAS,                                     X
               STORCLAS=STORCLAS,                                     X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
STRNAME  DC    CL26'LOG.STREAM.NAME'  stream name
DATACLAS DC    CL8'NEWCLASS'          data class name
MGMTCLAS DC    CL8'NEWMGMNT'          management class name
STORCLAS DC    CL8'NEWSTOR'           storage class name
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
RETCODE  DS    F                      return code from logger
RSNCODE  DS    F                      reason code from logger
DATAREA  DSECT
         IXGANSAA LIST=YES               answer area
```

Example 4

Issue IXGINVNT to define a log stream with a resource manager associated with it.

```
IXGINVNT REQUEST=DEFINE,
      TYPE=LOGSTREAM,
      STREAMNAME=SNAME,
      STRUCTNAME=STRUCT,
      RMNAME=RMNAME,
      STG_DUPLEX=NO,
      DESCRIPTION=DESCR,
      ANSAREA=XANSAREA,
      ANSLEN=XANSLEN,
      RSNCODE=RSCODE
*
SNAME    DS    CL26                    Stream name
STRUCT   DS    CL16                    Structure name
RMNAME   DS    CL8                     Res Man name
```

```
DESCR    DS    CL16                    Description
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 5

Issue IXGINVNT to define a log stream with a no retention period and autodeletion. This means that log data is deleted whenever IXGDELET is issued against the log stream.

```
         IXGINVNT REQUEST=DEFINE,
               TYPE=LOGSTREAM,
               STREAMNAME=SNAME,
               STRUCTNAME=STRUCT,
               STG_DUPLEX=NO,
               RETPD=0,AUTODELETE=YES,
               ANSAREA=XANSAREA,
               ANSLEN=XANSLEN,
               RSNCODE=RSCODE
SNAME    DS    CL26                    Stream name
STRUCT   DS    CL16                    Structure name
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 6

Issue IXGINVNT to define a log stream with staging data sets and a policy of unconditional duplexing. This means that data will always be duplexed to staging data sets, even if the configuration is not volatile.

```
IXGINVNT REQUEST=DEFINE,
      TYPE=LOGSTREAM,
      STREAMNAME=SNAME,
      STRUCTNAME=STRUCT,
      STG_DUPLEX=YES,DUPLEXMODE=UNCOND,
      ANSAREA=XANSAREA,
      ANSLEN=XANSLEN,
      RSNCODE=RSCODE
SNAME    DS    CL26                    Stream name
STRUCT   DS    CL16                    Structure name
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 7

Issue IXGINVNT REQUEST=DELETE to delete a structure definition.

```
          IXGINVNT REQUEST=DELETE,                                  X
               TYPE=STRUCTURE,                                      X
               STRUCTNAME=STRUCT,                                   X
               ANSAREA=ANSAREA,                                     X
               ANSLEN=ANSLEN,                                       X
               RSNCODE=RSNCODE,                                     X
               MF=S,                                                X
               RETCODE=RETCODE
STRUCT   DC    CL16'LIST01'           structure name
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
RETCODE  DS    F                      return code from logger
```

```
RSNCODE  DS    F                      reason code from logger
DATAREA  DSECT
         IXGANSAA LIST=YES               answer area
```

Example 8

Issue IXGINVNT with in list, execute and modify forms.

```
          IXGINVNT MF=(L,IXGINVNT_PLIST)
          IXGINVNT REQUEST=DEFINE,                                    X
               STREAMNAME=STRNAME,                                    X
               MF=(M,IXGINVNT_PLIST,NOCHECK)

          IXGINVNT REQUEST=DEFINE,                                    X
               TYPE=LOGSTREAM,                                        X
               MODEL=NO,                                              X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=(E,IXGINVNT_PLIST,NOCHECK)                          X
               RETCODE=RETCODE
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
STRNAME  DC    CL26'LOG.STREAM.NAME'  stream name
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 9

Issue IXGINVNT using registers.

```
          LA    R6,STRUCT              load struture name into reg 6
          IXGINVNT REQUEST=DELETE,                                    X
               TYPE=STRUCTURE,                                        X
               STRUCTNAME=(6),                                        X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
STRUCT   DC    CL16'LIST01'           structure name
ANSAREA  DS    CL(ANSAA_LEN)          answer area for log requests
ANSLEN   DC    A(L'ANSAREA)           length of logger's answer area
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 10

Issue IXGINVNT REQUEST=DEFINE to define a log stream as DASD-only:

```
          IXGINVNT REQUEST=DEFINE,                                    X
               TYPE=LOGSTREAM,                                        X
               STREAMNAME=STRNAME,                                    X
               DASDONLY=YES,                                          X
               MAXBUFSIZE=MAXBUF,                                     X
               HLQ=HLQ,                                               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
STRNAME  DC    CL26'LOG.STREAM.NAME'  log stream name
```

```
MAXBUF   DC    F'65532'               maximum buffer size
HLQ      DC    CL8'USERNAME'          high level qualifier
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 11

Issue IXGINVNT REQUEST=DEFINE to define a log stream as DASD-only and then issue the IXGINVNT REQUEST=UPDATE request to upgrade the DASD-only log stream to a coupling facility log stream, associating it with structure 1:

```
          IXGINVNT REQUEST=DEFINE,                                    X
               TYPE=LOGSTREAM,                                        X
               STREAMNAME=STRNAME,                                    X
               DASDONLY=YES,                                          X
               MAXBUFSIZE=MAXBUF,                                     X
               HLQ=HLQ,                                               X
               ANSAREA=ANSAREA,                                       X
               ANSLEN=ANSLEN,                                         X
               RSNCODE=RSNCODE,                                       X
               MF=S,                                                  X
               RETCODE=RETCODE
          IXGINVNT REQUEST=UPDATE,                                    X
               TYPE=LOGSTREAM,                                        X
               STREAMNAME=STRNAME,                                    X
               STRUCTNAME=STRUCT,                                     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
STRNAME  DC    CL26'LOG.STREAM.NAME'  log stream name
STRUCT   DC    CL16'STRUCTURE1'       structure name
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
```

IXGOFFLD — Initiate Offload to DASD Log Data Sets

Description

The IXGOFFLD macro allows the caller to intiate an offload of log data from the coupling facility structure for coupling facility log streams and from local storage buffers for DASD-only log streams to DASD log data sets.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state. Any PSW key
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 for I/O and external interrupts The caller's parameter list must be resident in the caller's primary address space. All storage areas specified must be in the same storage key as the caller.
Locks:	No locks may be held.
Control parameters:	None.

Programming Requirements

- Before issuing this request, the caller must have issed IXGCONN to connect to the log stream. The caller must specify specify AUTH=WRITE on the IXGCONN request.
- The current primary address space must be the same as the HOME address space at the time you issued the IXGCONN macro.
- 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 must exist in the caller's primary address space.

Input Register Information

Before issuing the IXGOFFLD 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:

© Copyright IBM Corp. 1988, 2002

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

IBM recommends that you use IXGOFFLD only when essential. The offloading process does entail some overhead and may degrade system logger performance.

Syntax

The IXGOFFLD macro is written as follows:

---

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

STREAMTOKEN=*streamtoken*	*streamtoken*: 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).
,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**	**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)	*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 IXGOFFLD 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 on the IXGCONN service.

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

,ANSAREA=*ansarea*
A required input parameter of a virtual storage area, called the answer area. The **ANSAREA** contains additional error status when the IXGOFFLD service generates an error return code. The format of the returned data is defined 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 contains the length in bytes of the virtual storage area provided for **ANSAREA**.

The length 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 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**, which supports all parameters except those 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

1C5 **Ixg_Abend_Code -** A System Logger abend has occurred.

Reason Code (Hex)	Explanation
xxxx085F	**IxgRsnCodePercToRequestor -** **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.

Return and Reason Codes

When the IXGOFFLD 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.

00 IxgRetCodeOk - Successful Completion

04 IxgRetCodeWarning - The request was processed successfully, however a warning condition was encountered.

08 IxgRetCodeError - An error has been encountered. The associated reason code provides more information.

0C IxgRetCodeCompError - A System Logger component error has been encountered.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 21. Return and Reason Codes for the IXGOFFLD Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**IxgRsnCodeOk -** **Explanation:** Request processed successfully.
08	xxxx0801	**IxgRsnCodeBadParmlist -** **Explanation:** Program error. The parameter list is not valid. Either the parameter list storage is inaccessible, or the version of the macro used was not valid. **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:** In the answer area mapped by IXGANSAA, see ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.
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 connector's address space. **Action:** Do one of the following: • Make sure that the stream token specified is valid. • Ensure that IXGOFFLD requests were issued from the connector's address space.
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.

Table 21. Return and Reason Codes for the IXGOFFLD Macro (continued)

Return Code	Reason Code	Meaning and Action
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 might occur after the system logger address space has terminated. **Action:** Specify storage that is in the callers 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	xxxx0819	**IxgRsnCodeSRBMode -** **Explanation:** Program error. The calling program is in SRB mode, but task mode is the required dispatchable unit mode for this system logger service. **Action:** Make sure the calling program is in task mode.
08	xxxx081C	**IxgRsnCodeNotAuthFunc -** **Explanation:** Program error. The program connected to the log stream with the AUTH=READ parameter and then tried to delete, write, offload or update data. You cannot write, delete, update or offload data when connected with read authority. **Action:** Issue the IXGCONN service with AUTH=WRITE authority and then reissue this request.
08	xxxx082D	**IxgRsnCodeExpiredStmToken -** **Explanation:** Environment error. The stream token is no longer valid because the connector has been disconnected. **Action:** Reconnect to the logstream before issuing any functional requests.
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.

Table 21. Return and Reason Codes for the IXGOFFLD Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0861	**IxgRsnCodeRebuildInProgress -** **Explanation:** Environment error. No requests can be processed for this log stream because a coupling facility structure rebuild is in progress for the structure associated with this log stream. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0862	**IxgRsnCodeXESPurge -** **Explanation:** Environment error. An cross-system extended services (XES) request has been purged due to rebuild processing. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0863	**IxgRsnCodeStructureFailed -** **Explanation:** Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
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 rebuild the log stream in another coupling facility or the log stream will be disconnected. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available. • The log stream has been disconnected from this system.
08	xxxx0890	**IxgRsnCodeAddrSpaceNotAvail -** **Explanation:** System error. The system logger address space failed and is not available. **Action:** Do not issue system logger requests.

Table 21. Return and Reason Codes for the IXGOFFLD Macro (continued)

Return Code	Reason Code	Meaning and Action
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. When it's available, reconnect 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	xxxx08DF	**IxgRsnCodeOffLoadFlushError -** **Explanation:** System error. The flush service called by IXGOFFLD encountered a XES error. **Action:** Examine the answer area, which contains more detailed information about the error.
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

Issue IXGOFFLD to initiate offload processing for a log stream.

```
         IXGOFFLD                                                  @
               STREAMTOKEN=OTOKEN,                                 @
               ANSAREA=XANSAREA,                                   @
               ANSLEN=XANSLEN,                                     @
               RSNCODE=RSCODE
OTOKEN   DS    CL16                   Output 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
```

IXGQUERY — Query a Log Stream for Information

Description

The IXGQUERY macro allows a user to retrieve information about a log stream.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state. Any PSW key
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 for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	None.

Programming Requirements

- The caller must have a valid connection to the log stream.
- The current primary address space must be the same as the HOME address space at the time you issued the IXGCONN macro.
- 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.
- Include mapping macro IXGBQBUF in your program. This macro shows the format of the data returned by IXGQUERY.

Restrictions

- The caller's buffer must be in the caller's primary address space and cannot be ALET-qualified.
- All storage areas specified must be in the same storage key as the caller. Storage areas must exist in the caller's primary address space.
- The caller cannot have any enabled, unlocked task (EUT) FRRs established.

Input Register Information

Before issuing the IXGQUERY 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

© Copyright IBM Corp. 1988, 2002

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 IXGQUERY macro is written as follows:

---

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

STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or address in register (2) - (12).
,BUFFER=*buffer*	*buffer*: RS-type address or address in register (2) - (12).
,BUFFLEN=*bufflen*	*bufflen*: 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).
,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**	**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)	*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 IXGQUERY 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 output parameter that specifies the buffer into which the requested data are to be copied. The contents of the buffer are mapped by IXGQBUF.

The buffer cannot be ALET qualified.

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

,BUFFLEN=*bufflen*
A required input parameter that specifies the length of the buffer identified by the **BUFFER** keyword.

If the user-specified buffer is not large enough to return the specified data, a specific return/reason code will be returned.

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

,ANSAREA=*ansarea*
A required input parameter of a virtual storage area, called the answer area. The **ANSAREA** contains additional error status when the IXGQUERY service generates an error return code. The format of the returned data is defined 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 contains the length in bytes of the virtual storage area provided for **ANSAREA**.

The length 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 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**, which supports all parameters except those 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

The IXGQUERY service can issue abend X'1C5' with reason code X'0805'. This abend indicates an abend during system logger processing. If you receive this abend, reissue the request. If the problem persists, contact the IBM Support Center.

Return and Reason Codes

When the IXGQUERY 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.

00 IxgRetCodeOk - Successful Completion

04 IxgRetCodeWarning - The request was processed successfully, however a warning condition was encountered.

08 IxgRetCodeError - An error has been encountered. The associated reason code provides more information.

0C IxgRetCodeCompError - A System Logger component error has been encountered.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 22. Return and Reason Codes for the IXGQUERY Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**IxgRsnCodeOk -** **Explanation:** Request processed successfully.
08	xxxx0801	**IxgRsnCodeBadParmlist -** **Explanation:** Program error. The parameter list is not valid. Either the parameter list storage is inaccessible, or the version of the macro used was not valid. **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:** In the answer area mapped by IXGANSAA, see ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.
08	xxxx0803	**IxgRsnCodeBadBuffer -** **Explanation:** The virtual storage area specified by the BUFFER keyword not addressable. **Action:** Ensure the storage area is accessible to the Logger Services for the duration of the request.
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 IXGQUERY requests were issued from the connectors address space.
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	xxxx080F	**IxgRsnCodeBadBufsize -** **Explanation:** The BUFFER specified is not large enough to contain the data being returned. No data is returned. **Action:** Obtain a buffer of the length of IXGQBUF and redrive the request.

Table 22. Return and Reason Codes for the IXGQUERY Macro (continued)

Return Code	Reason Code	Meaning and Action
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	**IxgRsnCodeBadAnslen -** **Explanation:** Program error. The answer area length (ANSLEN parameter) is not large enough. The system logger returned the required size in the Ansaa_Preferred_Size field of the answer area, mapped by IXGANSAA macro. **Action:** Reissue the request, specifying an answer area of the required size.
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 callers 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	xxxx0819	**IxgRsnCodeSRBMode -** **Explanation:** Program error. The calling program is in SRB mode, but task mode is the required dispatchable unit mode for this system logger service. **Action:** Make sure the calling program is in task mode.
08	xxxx082D	**IxgRsnCodeExpiredStmToken -** **Explanation:** Environment error. The stream token is no longer valid because the connector has been disconnected. **Action:** Reconnect to the logstream before issuing any functional requests.
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.

Table 22. Return and Reason Codes for the IXGQUERY Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0861	**IxgRsnCodeRebuildInProgress -** **Explanation:** Environment error. No requests can be processed for this log stream because a coupling facility structure rebuild is in progress for the structure associated with this log stream. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0862	**IxgRsnCodeXESPurge -** **Explanation:** Environment error. An cross-system extended services (XES) request has been purged due to rebuild processing. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0863	**IxgRsnCodeStructureFailed -** **Explanation:** Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
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 rebuild the log stream in another coupling facility or the log stream will be disconnected. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available. • The log stream has been disconnected from this system.
08	xxxx0890	**IxgRsnCodeAddrSpaceNotAvail -** **Explanation:** System error. The system logger address space failed and is not available. **Action:** Do not issue system logger requests.

Table 22. Return and Reason Codes for the IXGQUERY Macro (continued)

Return Code	Reason Code	Meaning and Action
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, reconnect 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	xxxx08D3	**IxgRsnCodeFuncNotSupported -** **Explanation:** Environment error. The query request failed because the LOGR couple data set is not at the correct level. The inventory must be at least at the OS390R3 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

Issue IXQUERY to get information about a log stream.

```
         IXGQUERY                                                       @
               STREAMTOKEN=OTOKEN,                                      @
               BUFFER=QRYBUFF,                                          @
               BUFFLEN=QRYBUFF_LEN,                                     @
               ANSAREA=XANSAREA,                                        @
               ANSLEN=XANSLEN,                                          @
               RSNCODE=RSCODE
OTOKEN   DS    CL16                    Output Stream token
QRYBUFF  DS    CL(QBUF_LEN)            IXGQUERY data area
QRYBUFF_LEN DC A(QBUF_LEN)             IXGQUERY data length
XANSAREA DS    CL(ANSAA_LEN)           Logger answer area
XANSLEN  DC    A(ANSAA_LEN)            Answer area length
RSCODE   DS    F                       Reason code
         DSECT  ,
         IXGQBUF  ,                    The macro for IXGQUERY data
         IXGANSAA ,                    The answer area macro
```

IXGUPDAT — Update Log Stream Control Information

Description

The IXGUPDAT macro allows the caller to update the GMT time stamp maintained in the control information for a log stream. When this field is successfully updated, any future log blocks written to the log stream cannot will have a time stamp less than the updated time stamp. (Note that this service does not affect time stamps that the application imbeds in the log block.)

Environment

The requirements for the caller are:

Minimum authorization:	Problem state. Any PSW key
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 for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	None.

Programming Requirements

- The caller must have a valid connection to the target log stream, specifying AUTH=WRITE.
- 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.
- The current primary address space must be the same as the HOME address space at the time you issued the IXGCONN macro.

Restrictions

All storage areas specified must be in the same storage key as the caller. Storage areas that must exist in the caller's primary address space.

Input Register Information

Before issuing the IXGUPDAT 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

© 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 IXGUPDAT macro is written as follows:

---

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

STREAMTOKEN=*streamtoken*	*streamtoken*: RS-type address or address in register (2) - (12).
,GMT_TIMESTAMP=*gmt_timestamp*	*gmt_timestamp*: RS-type address or address in register (2) - (12).
,GMT_TIMESTAMP=**NO_GMT_TIMESTAMP**	**Default**: GMT_TIMESTAMP=NO_GMT_TIMESTAMP
,ANSAREA=*ansarea*	*ansarea*: RS-type address or address in register (2) - (12).
,ANSLEN=*anslen*	*anslen*: 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**)	**Default**: MF=S *list addr*: RS-type address or register (1) - (12).

,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 IXGUPDAT 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 on the IXGCONN service.

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

,GMT_TIMESTAMP=*gmt_timestamp*
,GMT_TIMESTAMP=NO_GMT_TIMESTAMP
An optional input parameter that lets you modify the GMT time stamp in the coupling facility structure list controls. You must supply a time stamp that is equal to or greater than the current time stamp maintained in the Log Stream Control information. Once modified, the next log blocks written to the log stream will be assigned a GMT time stamp equal to or greater than the one specified on the IXGUPDAT request. The default is NO_GMT_TIMESTAMP.

If NO_Gmt_TimeStamp is specified for GMT_TimeStamp the macro will be invoked as if GMT_TimeStamp was not specified.

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

,ANSAREA=*ansarea*
A required input parameter of a virtual storage area, called the answer area. The **ANSAREA** contains additional error status when the IXGUPDAT service generates an error return code. The format of the returned data is defined 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 contains the length in bytes of the virtual storage area provided for **ANSAREA**.

The length 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 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**, which supports all parameters except those 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

The IXGQUERY service can issue abend X'1C5' with reason code X'085F'. This abend indicates an abend during system logger processing. If you receive this abend, reissue the request. If the problem persists, contact the IBM Support Center.

Return and Reason Codes

When the IXGUPDAT 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.

00 IxgRetCodeOk - Successful Completion

04 IxgRetCodeWarning - The request was processed successfully, however a warning condition was encountered.

08 IxgRetCodeError - An error has been encountered. The associated reason code provides more information.

0C IxgRetCodeCompError - A System Logger component error has been encountered.

The following table contains hexadecimal return and reason codes, the equate symbols associated with each reason code, and the meaning and suggested action for each return and reason code.

Table 23. Return and Reason Codes for the IXGUPDAT Macro

Return Code	Reason Code	Meaning and Action
00	xxxx0000	**IxgRsnCodeOk -** **Explanation:** Request processed successfully.
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	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 IXGUPDAT requests were issued from the connectors address space.
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.

Table 23. Return and Reason Codes for the IXGUPDAT Macro (continued)

Return Code	Reason Code	Meaning and Action
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 callers 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	xxxx0819	**IxgRsnCodeSRBMode -** **Explanation:** Program error. The calling program is in SRB mode, but task mode is the required dispatchable unit mode for this system logger service. **Action:** Make sure the calling program is in task mode.
08	xxxx081C	**IxgRsnCodeNotAuthFunc -** **Explanation:** Program error. The program connected to the log stream with the AUTH=READ parameter and then tried to delete, write, offload or update data. You cannot write, delete, offload or update data when connected with read authority. **Action:** Issue the IXGCONN service with AUTH=WRITE authority and then reissue this request.
08	xxxx082D	**IxgRsnCodeExpiredStmToken -** **Explanation:** Environment error. The stream token is no longer valid because the connector has been disconnected. **Action:** Reconnect to the logstream before issuing any functional requests.
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.

Table 23. Return and Reason Codes for the IXGUPDAT Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0861	**IxgRsnCodeRebuildInProgress -** **Explanation:** Environment error. No requests can be processed for this log stream because a coupling facility structure rebuild is in progress for the structure associated with this log stream. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0862	**IxgRsnCodeXESPurge -** **Explanation:** Environment error. An cross-system extended services (XES) request has been purged due to rebuild processing. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
08	xxxx0863	**IxgRsnCodeStructureFailed -** **Explanation:** Environment error. Either the coupling facility structure associated with the log stream has failed or the coupling facility itself has failed. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available.
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 rebuild the log stream in another coupling facility or the log stream will be disconnected. **Action:** Listen for ENF signal 48 that will indicate one of the following: • The log stream is available because the rebuild completed successfully. Reissue the request. • The rebuild failed and the log stream is not available. • The log stream has been disconnected from this system.
08	xxxx0890	**IxgRsnCodeAddrSpaceNotAvail -** **Explanation:** System error. The system logger address space failed and is not available. **Action:** Do not issue system logger requests.

Table 23. Return and Reason Codes for the IXGUPDAT Macro (continued)

Return Code	Reason Code	Meaning and Action
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, reconnect 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	xxxx08DD	**IxgRsnCodeUpdateTimeStampTooSmall -** **Explanation:** Program error. The replacement GMT time stamp is smaller than the time stamp maintained in the coupling facility for the log stream. This error can be caused because the application did in fact specify an invalid time stamp or the time stamp value has changed after its current value was retrieved (e.g., via the IXGQUERY service) because a write or another update request was successfully processed for the log stream somewhere in the sysplex. **Action:** Invoke the IXGQUERY service to obtain the current time stamp value and determine if the update request should be retried.
08	xxxx08DE	**IxgRsnCodeUpdateNoOptions -** **Explanation:** Program error. The IXGUPDAT macro was invoked with no options specified. **Action -** - Specify at least one option and retry the request.
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

Issue IXGUPDAT to update the time stamp for a log stream.

```
         IXGUPDAT                                                  @
               STREAMTOKEN=OTOKEN,                                 @
               GMT_TIMESTAMP=GMTTIME,                              @
               ANSAREA=XANSAREA,                                   @
               ANSLEN=XANSLEN,                                     @
               RSNCODE=RSCODE
OTOKEN   DS    CL16                   Output Stream token
GMTTIME  DS    CL8                    GMT
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
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).
,TIMESTAMP=*timestamp*	*timestamp*: RS-type address or register (2) - (12). **Default**: NO_TIMESTAMP
MODE=SYNC MODE=ASYNCNORESPONSE MODE=SYNCECB	**Default**: MODE=SYNC
,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) ,MF=(M,*list addr*,NOCHECK)	**Default**: MF=S

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.

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
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.

,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 24. 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.

Table 24. Return and Reason Codes for the IXGWRITE Macro (continued)

Return Code	Reason Code	Meaning and Action
04	xxxx0407	**Equate Symbol**: IxgRsnCodeConnPossibleLossOfData **Explanation**: Environment error. The request was successful, but there may be log blocks permanently missing between this log block and the one previously returned. This condition occurs when a system or coupling facility fails and not all of the data in the log stream could be recovered. **Action**: If your application cannot tolerate any data loss, stop issuing system logger services to this log stream, disconnect from the log stream, and reconnect to a new, undamaged log stream. You can continue using the log stream if your applications can tolerate data loss.
04	xxxx0408	**Equate Symbol**: IxgRsnCodeDsDirectoryFullWarning **Explanation**: Environment error. The request was successful, but the log streams DASD data set directory is full. System logger cannot offload any further data from the coupling facility structure to DASD. The system logger will continue to process IXGWRITE requests until this log streams portion of the coupling facility structure becomes full. **Action**: Either delete enough data from the log stream to free up space in the log streams data set directory so that offloading can occur or disconnect from the log stream.
04	xxxx0409	**Equate Symbol**: IxgRsnCodeWowWarning **Explanation**: Environment error. The request was successful, but an error condition was detected during a previous offload of data. System logger might not be able to offload further data. System logger will continue to process IXGWRITE requests only until the interim storage for the log stream is filled. (Interim storage is the coupling facility for a coupling facility log stream and local storage buffers for a DASD-only log stream.) **Action**: Do not issue any further requests for this log stream and disconnect. Connect to another log stream. Check the system log for message IXG301I to determine the cause of the error. If you cannot fix the error, search problem reporting data bases for a fix for the problem. If no fix exists, contact the IBM Support Center.
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.

Table 24. Return and Reason Codes for the IXGWRITE Macro (continued)

Return Code	Reason Code	Meaning and Action
08	xxxx0801	**Equate Symbol**: IxgRsnCodeBadParmlist **Explanation**: Program error. The parameter list could not be accessed. **Action**: Ensure that the storage area for the parameter list is accessible to the system logger for the duration of the request. The parameter list storage must be addressable in the caller's primary address space and in the same key as the caller.
08	xxxx0802	**Equate Symbol**: IxgRsnCodeXESError **Explanation**: System error. A severe cross-system extended services (XES) error has occurred. **Action**: See ANSAA_DIAG1 for the XES return code and ANSAA_DIAG2 for the XES reason code.
08	xxxx0803	**Equate Symbol**: IxgRsnCodeBadBuffer **Explanation**: Program error. The virtual storage area specified on the BUFFER 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.

Table 24. Return and Reason Codes for the IXGWRITE 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	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.

Table 24. Return and Reason Codes for the IXGWRITE 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	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).

Table 24. Return and Reason Codes for the IXGWRITE Macro (continued)

Return Code	Reason Code	Meaning and Action
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.
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.

Table 24. Return and Reason Codes for the IXGWRITE 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	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.

Table 24. Return and Reason Codes for the IXGWRITE 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 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.
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.

Table 24. Return and Reason Codes for the IXGWRITE 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.

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
```

LINK and LINKX — Pass Control to a Program in Another Load Module

Description

The LINK macro is used to pass control to a specified entry name in another load module; the entry name must be a member name or an alias in the directory of a partitioned data set (PDS) or must have been specified in an IDENTIFY macro. The load module containing the program is brought into virtual storage if a usable copy is not available.

If your program is in access register (AR) address space control (ASC) mode, use LINKX. All the parameters on LINK are valid on LINKX.

Descriptions of the LINK and LINKX macro in this book are:

- The standard form of the LINK macro, which includes general information about the LINK and LINKX macros with specific information about the LINK macro. The syntax of the LINK macro and all LINK parameters are explained.
- The standard form of the LINKX macro, which presents information specific to the LINKX macro and callers in AR mode.
- The list form of the LINK and LINKX macros.
- The execute form of the LINK and LINKX macros.

LINK and LINKX processing ensure that the called program receives control in the correct addressing mode. If the called program has an address mode of ANY, it receives control in the AMODE of the calling program. The program issuing the LINK or LINKX macro regains control in its own addressing mode.

The caller optionally can provide a parameter list to be passed to the called program. If the called program terminates abnormally, or if the specified entry point cannot be located, the task is abnormally terminated unless the caller provides an ERRET exit.

Note

The LINK and LINKX macros have the same environment specifications, register information, programming requirements, restrictions and limitations, performance implications, and return and reason codes described below, except where noted in the explanation for LINKX.

Environment

The requirements for the caller of LINK are:

Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=SASN=HASN
AMODE:	24- or 31-bit for LINK. 24- or 31- or 64-bit for LINKX.
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.

© Copyright IBM Corp. 1988, 2002

Programming Requirements

None.

Restrictions

- The caller cannot have an EUT FRR established.

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.

If the LINK is successful, the GPRs contain the following when the **called program** receives control:

Register	Contents
0	One of the following: • Used as a work register by the system if SF is specified. • Otherwise, unchanged.
1	One of the following: • Address of the PARAM address list if that is coded. • Otherwise, unchanged if LSEARCH=YES not specified and LINKX not specified, and LINK not issued with SYSSTATE ASCENV=AR. • Otherwise, used as a work register by the system.
2-13	Unchanged
14	Used as a work register by the system
15	Requested program's entry point address

Upon return to the caller, the GPRs contain whatever values the called program placed there.

If the LINK is not successful and the caller provided an ERRET exit to receive control, the GPRs contain the following:

Register	Contents
0	One of the following: • Used as a work register by the system if SF is specified. • Otherwise, unchanged.
1	Abend code for the ABEND that would have been issued if the caller had not provided an ERRET exit.
2-13	Unchanged
14	Used as a work register by the system
15	Address of the ERRET exit.

Performance Implications

None.

Syntax

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

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

EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address, or register (2) - (12).
DE=*list entry addr*	*list entry addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,PARAM=(*addr*) ,PARAM=(*addr*),VL=1	*addr*: A-type address, or register (2) - (12). **Note**: *addr* is one or more addresses, separated by commas. For example, (*addr,addr,addr*)
,ID=*id nmbr*	*id nmbr*: Symbol or decimal digit, with a maximum value of 4095.
,ERRET=*err rtn addr*	*err rtn addr*: A-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: No

Parameters

The parameters are explained as follows:

EP=*entry name*
EPLOC=*entry name addr*
DE=*list entry addr*

Specifies the entry name, the address of the entry name, or the address of the name field in a 62-byte list entry for the entry name that was constructed using the BLDL macro. If EPLOC is coded, *entry name addr* points to an eight-byte field. If the name is less than eight characters, left-justify the name and pad with blanks on the right to make up the eight characters.

The system ignores the information you specify on the DE parameter if the parameter does one or both of the following:

- Specifies an entry in an authorized library (that is, defined in IEAAPFxx member of parmlib)
- Requests access to a program or library that is controlled by the system authorization facility (SAF)

Instead, the system uses the BLDL macro to construct a new list entry containing the DE information.

Note: When you use the DE parameter with the LINK macro, DE specifies the address of a list that was created by a BLDL macro. BLDL and LINK must be issued from the same task; otherwise, the system might terminate the program with an abend code of 106 and a return code of 15. Therefore, do not issue ATTACH or DETACH between issuances of BLDL and LINK.

,DCB=*dcb addr*
Specifies the address of the opened data control block for the partitioned data set containing the entry name described above. This parameter must indicate the same DCB specified in the BLDL used to locate the entry name.

If the DCB parameter is omitted or if DCB=0 is specified when the LINK macro is issued by the *job step task,* the data sets referred to by either the STEPLIB or JOBLIB DD statement are first searched for the entry point name. If the entry point name is not found, the link library is searched.

If the DCB parameter is omitted or if DCB=0 is specified when the LINK macro is issued by a *subtask*, the data sets associated with one or more data control blocks referred to by the TASKLIB operand of previous ATTACH macros in the subtasking chain are first searched for the entry point name. If the entry point name is not found, the search is continued as if LINK had been issued by the job step task.

Note: DCB must reside in 24-bit addressable storage.

,PARAM=(*addr***)**
,PARAM=(*addr***),VL=1**
Specifies address(es) to be passed to the called program. To form the parameter list, the macro expands each address inline to a fullword on a fullword boundary, in the order designated. GPR 1 contains the address of the first parameter when the program is given control. (If this parameter is not coded, GPR 1 is not altered unless the execute form of the LINK macro is coded or LSEARCH=YES is specified.)

Specify VL=1 only if the called program can be passed a variable number of parameters. VL=1 causes the high-order bit of the last address parameter to be set to 1; the bit can be checked to find the end of the list.

Note: If you specify only one address for PARAM=, you do not need to enter the parentheses.

,ID=*id nmbr*
Specifies an identifier for this invocation of the macro, useful for debugging purposes. The last fullword of the macro expansion is a NOP instruction containing, in bytes 3 and 4, the identifier you specified.

,ERRET=*err rtn addr*
Specifies the address of an exit to receive control when an error condition that would cause abnormal termination of the task is detected. The ERRET exit does not receive control when input parameter errors are detected.

,LSEARCH=NO
,LSEARCH=YES
Specifies whether (YES) or not (NO) the search is to be limited to the job pack area and the first library in the normal search sequence.

Return and Reason Codes

None.

Example 1

Pass control to a specified entry name (PGMLKRUS) in another load module. Let the system find the module from available libraries.

```
LINK    EP=PGMLKRUS
```

Example 2

Pass control to a specified entry name (PGMA) in another load module, specifying (in registers 4, 6, 8) three addresses to be passed to the called program.

```
LINK EP=PGMA,PARAM=((4),(6),(8))
```

LINKX — Pass Control to a Program in Another Load Module

The LINKX macro performs the same function as LINK. It passes control to a specified entry name in another load module. LINKX is intended for use by programs running in access register (AR) mode.

Note

The LINKX macro has the same environment specifications, register information, programming requirements, restrictions and limitations, performance implications, and return and reason codes as the LINK macro, except where noted below.

Environment

The LINKX macro can be used by callers in AR or primary ASC mode.

Programming Requirements

If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before you issue LINKX.

Parameters passed to the called program using the PARAM parameter must reside in your primary address space.

Register Information

When the caller regains control or the ERRET exit receives control, the access registers (ARs) are unchanged.

Syntax

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

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

EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address, or register (2) - (12).
DE=*list entry addr*	*list entry addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,PARAM=(*addr*) ,PARAM=(*addr*),VL=1	*addr*: A-type address, or register (2) - (12). **Note**: *addr* is one or more addresses, separated by commas. For example, (*addr,addr,addr*)
,PLIST4=YES, ,PLIST4=NO,	**Default**: None.
,PLIST8=YES, ,PLIST8=NO,	**Default**: None.
,ID=*id nmbr*	*id nmbr*: Symbol or decimal digit, with a maximum value of 4095.
,ERRET=*err rtn addr*	*err rtn addr*: A-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: NO
,AMODE64OK=NO ,AMODE64OK=YES	**Default**: NO

Parameters

The parameters are explained under LINK with the following exceptions. The parameter list on the PARAM parameter is different for callers in AR mode. It is described as follows:

PARAM=(*addr*)
PARAM=(*addr*),VL=1

Specifies addresses to be passed to the called program. The macro expands each address inline to a fullword on a fullword boundary, in the order designated.

LINKX builds the parameter list so that the addresses passed to the called program are in the first half of the parameter list and their corresponding ALETs are in the last half of the list.

When the program that is the target of the LINKX receives control, general purpose register 1 contains the address of the parameter list. If the program that issued the LINKX macro was in AR mode, access register 1 contains the ALET that qualifies the parameter list address.

Specify VL=1 if the called program can be passed a variable number of parameters. VL=1 causes the macro to set the high-order of the last address parameter to 1. For callers in AR mode, the ALETs follow this last address parameter. For more information about passing parameters in AR mode, see “User Parameters” on page 4.

Note: If you specify only one address for PARAM=, you do not need to enter the parentheses.

,PLIST4=YES
,PLIST4=NO

,PLIST8=YES
,PLIST8=NO

Defines the size of the parameter list entries for a parameter list to be built by LINKX based on the PARAM keyword.

PLIST4 and PLIST8 cannot be specified together. If neither is specified, the default is:

- If running AMODE 64, PLIST8=YES
- If not running AMODE 64, PLIST4=YES

If running AMODE 64 and PLIST4=YES is specified, the system builds a 4-bytes-per-entry parameter list just as it would if the program were running AMODE 24 or AMODE 31 and did not specify PLIST4 or PLIST8.

If running AMODE 24 or AMODE 31 and PLIST8 is specified, the system builds an 8-bytes-per-entry parameter list just as it would if the program were running AMODE 64 and did not specify PLIST4 or PLIST8.

,AMODE64OK=NO
,AMODE64OK=YES

Indicates if the system is to accept an attempt to link to an AMODE 64 target routine from an AMODE 24 or AMODE 31 routine.

NO
Indicates that the system is to abend such an attempt.

YES
Indicates that the system is to accept such an attempt.

LINK and LINKX—List Form

Two parameter lists are used in a LINK or LINKX macro: a control program parameter list and problem program parameter list. Only the control program parameter list can be constructed in the list form of LINK or LINKX. Address parameters to be passed in a parameter list to the problem program can be provided using the list form of CALL. This parameter list can be referred to in the execute form of LINK or LINKX.

Syntax

The list form of the LINK or LINKX macro is written as follows:

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

EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address.
DE=*list entry addr*	*list entry addr*: A-type address.
,DCB=*dcb addr*	*dcb addr*: A-type address.
,ERRET=*err rtn addr*	*err rtn addr*: A-type address.
,LSEARCH=NO ,LSEARCH=YES	**Default**: No
,AMODE64OK=NO ,AMODE64OK=YES	AMODE64OK is valid only with LINKX. **Default**: NO
,SF=L	

Parameters

The parameters are explained under the standard form of the LINK and LINKX macros, with the following exception:

,SF=L
Specifies the list form of the LINK or LINKX macro.

Notes:

1. Coding the LSEARCH parameter causes a parameter list to be created that is different from the list created when LSEARCH is omitted. If you code LSEARCH=YES in either the list or execute form of the macro, you must code it in both forms.
2. If ERRET is coded in the list form and not specified in the execute form, the error routine specified in the list form will be retained and used in the execute form of the macro. If ERRET is specified in both the list and the execute form, the error routine specified in the execute form of the macro will be used.

LINK and LINKX—Execute Form

Two parameter lists are used in a LINK or LINKX macro: a control program parameter list and an optional problem program parameter list. Either or both of these lists can be remote and can be referred to and modified by the execute form of LINK or LINKX. If only one of the parameter lists is remote, parameters that require use of the other parameter list cause that list to be constructed inline as part of the macro expansion.

Syntax

The execute form of the LINK or LINKX macro is written as follows:

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

b	One or more blanks must precede LINK or LINKX.
LINK LINKX	
b	One or more blanks must follow LINK or LINKX.

EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: RX-type address or register (2) - (12).
DE=*list entry addr*	*list entry addr*: RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: RX-type address, or register (2) - (12).
,PARAM=(*addr*) ,PARAM=(*addr*),VL=1	*addr*: RX-type address, or register (2) - (12). **Note**: *addr* is one or more addresses, separated by commas. For example, (*addr,addr,addr*)
,PLIST4=YES, ,PLIST4=NO,	PLIST4 is valid only with LINKX. **Default**: None.
,PLIST8=YES, ,PLIST8=NO,	PLIST8 is valid only with LINKX. **Default**: None.
,ID=*id nmbr*	*id nmbr*: Symbol or decimal digit, with a maximum value of 4095.
,ERRET=*err rtn addr*	*err rtn addr*: RX-type address or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: No
,AMODE64OK=NO ,AMODE64OK=YES	AMODE64OK is valid only with LINKX. **Default**: NO
,MF=(E,*prob addr*) ,SF=(E,*ctrl addr*) ,MF=(E,*prob addr*),SF=(E,*ctrl addr*)	*prob addr*: RX-type address, or register (1) or (2) - (12). *ctrl addr*: RX-type address, or register (2) - (12) or (15).

Parameters

The parameters are explained under the standard form of the LINK and LINKX macros, with the following exceptions:

,MF=(E,*prob addr*)
,SF=(E,*ctrl addr*)
,MF=(E,*prob addr*),SF=(E,*ctrl addr*)
Specifies the execute form of the LINK or LINKX macro. This form uses a remote problem program parameter list, a remote control program parameter list, or both.

Notes:

1. Coding the LSEARCH parameter causes a parameter list to be created that is different from the list created when LSEARCH is omitted. If you code LSEARCH=YES in either the list or execute form of the macro, you must code it in both forms.
2. If ERRET is coded in the list form and not specified in the execute form, the error routine specified in the list form will be retained and used in the execute form of the macro. If ERRET is specified in both the list and the execute form, the error routine specified in the execute form of the macro will be used.

LOAD — Bring a Load Module into Virtual Storage

Description

The LOAD macro is used to bring the load module containing the specified entry name into virtual storage, if a usable copy is not available in virtual storage. Control is *not* passed to the load module; instead, the load module's entry point address is returned in GPR 0. LOAD services places the load module in storage above or below 16 megabytes depending on the module's RMODE. The responsibility count for the load module is increased by one.

The load module remains in virtual storage until the responsibility count is reduced to 0 through task terminations or until the effects of all outstanding LOAD requests for the module have been canceled (using the DELETE macro), *and* there is no other requirement for the module.

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- 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

If you code the parameters LSEARCH or LOADPT, you will obtain a macro-generated parameter list. Therefore, except for the error routine address, all addresses must be specified as A-type addresses or registers (2) - (12).

Restrictions

- Any module loaded by a task will not be removed from virtual storage unless the task that loaded the module invokes the DELETE macro or terminates.
- The load module entry name must be listed as a member name or alias in a partitioned dataset directory or it must have been specified previously using the IDENTIFY macro. If the LOAD macro cannot find the specified entry name, the caller's task is abended unless the caller provides an ERRET exit.
- The caller cannot have an EUT FRR established.

Input Register Information

Before issuing the LOAD 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

If the LOAD is successful, the GPRs contain the following when control returns to the caller:

Register	Contents

© Copyright IBM Corp. 1988, 2002

0	Entry point address of the requested load module. Load services sets the high-order bit of the entry point address to the load module's AMODE. If the module's AMODE is ANY, it sets the indicator to the caller's AMODE.
1	The high-order byte contains the load module's APF authorization code. If the module's length value in doublewords is less than 16M (2**24) and the module does not have the RMODE(SPLIT) attribute, then the low-order three bytes contain the module length in doublewords. If the module's length value in doublewords is greater than or equal to 16M (2**24), the low-order three bytes contain zeros. To obtain the module length, issue the CSVQUERY macro with the OUTLENGTH parameter. If the module is a program object with the RMODE(SPLIT) attribute, the low-order three bytes contain zeros. To obtain the length and load point information for each segment, issue the CSVQUERY macro with the OUTXTLST parameter. When the module is a program object bound with the FETCHOPT=NOPACK option, the length value returned has been rounded to the fullpage-multiple area obtained with GETMAIN to hold the program object. If the program object is bound with the FETCHOPT=PACK option, the length value returned is the size indicated in the directory entry. See *z/OS DFSMS Program Management* for further information.
2-13	Unchanged.
14	Used as a work register by the system.
15	Zero, indicating successful completion.

If the LOAD is not successful and the caller provided an ERRET exit to receive control, the GPRs contain:

Register	Contents
0	Used as a work register by the system
1	System completion code for the abend that would have been issued had the caller not provided an ERRET exit
2-13	Unchanged
14	Used as a work register by the system
15	Reason code (never zero) associated with the system completion code contained in GPR 1

When control returns to the caller or the ERRET exit receives control, the access registers (ARs) are 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 LOAD macro is written as follows:

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

EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: If LSEARCH or LOADPT is specified, A-type address or register (2) - (12); otherwise, RX-type address or register (0) or (2) - (12).
DE=*list entry addr*	*list entry addr*: If LSEARCH or LOADPT is specified, A-type address or register (2) - (12); otherwise, RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: If LSEARCH or LOADPT is specified, A-type address or register (2) - (12); otherwise, RX-type address, or register (1) or (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr*: RX-type address or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: NO
,LOADPT=*addr*	*addr*: A-type address or register (2) - (12).
,EXTINFO=*addr*	*addr*: A-type address or register (2) - (12).
,RELATED=*value*	

Parameters

The parameters are explained as follows:

EP=*entry name*
EPLOC=*entry name addr*
DE=*list entry addr*

Specifies the entry name, the address of the name, or the address of the name field in a 62-byte list entry for the entry name that was constructed using the BLDL macro. If EPLOC is coded, the name must be padded to eight bytes, if necessary.

The system ignores the information you specify on the DE parameter if the parameter does one or both of the following:

- Specifies an entry in an authorized library (that is, defined in IEAAPFxx member of parmlib)

- Requests access to a program or library that is controlled by the system authorization facility (SAF)

Instead, the system uses the BLDL macro to construct a new list entry containing the DE information.

Note: When you use the DE parameter with the LOAD macro, DE specifies the address of a list that was created by a BLDL macro. BLDL and LOAD must be issued from the same task; otherwise, the system might terminate the program with an abend code of 106 and a return code of 15. Therefore, do not issue an ATTACH or a DETACH macro between issuances of the BLDL and the LOAD macros.

,DCB=*dcb addr*
Specifies the address of the opened data control block for the partitioned data set containing the entry name described above. This parameter must indicate the same DCB specified in the BLDL used to locate the entry name.

If the DCB parameter is omitted or if DCB=0 is specified when the LOAD macro is issued by the *job step task*, the data sets referred to by either the STEPLIB or JOBLIB DD statement are first searched for the entry name. If the entry name is not found, the link library is searched.

If the DCB parameter is omitted or if DCB=0 is specified when the LOAD macro is issued by a *subtask*, the data sets associated with one or more data control blocks referred to by the TASKLIB operand of previous ATTACH macro in the subtasking chain are first searched for the entry name. If the entry name is not found, the search is continued as if the LOAD had been issued by the job step task.

Note: DCB must reside in 24-bit addressable storage.

,ERRET=*err rtn addr*
Specifies the address of a routine to receive control when an error condition that would cause an abnormal termination of the task is detected. Register 1 contains the abend code that would have resulted had the task abended, and register 15 contains the reason code that is associated with the abend. The routine does not receive control when input parameter errors are detected.

,LSEARCH=NO
,LSEARCH=YES
Specifies whether (YES) or not (NO) the search is to be limited to the job pack area and the first library in the normal search sequence.

,LOADPT=*addr*
Specifies that the starting address at which the module was loaded is to be returned to the caller at the indicated address.

,EXTINFO=*addr*
Specifies a 304-byte area which upon return is to contain extended information. This area is mapped by dsect EXTI within macro CSVEXTI. Included in this area are :

- the extent list (each entry is mapped by dsect EXTIXE within macro CSVEXTI)
- the authorization code
- the entry point address

By using the EXTINFO keyword you can avoid the need to call CSVQUERY after doing the LOAD to obtain information that would not otherwise be returned

by LOAD. For example, if a program object length were greater than 128 megabytes or had been bound with RMODE=SPLIT, LOAD would not otherwise return the length information.

,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.

The RELATED parameter is available on macros that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE), and on macros that relate to previous occurrences of the same macros (for example, CHAP and ESTAE).

The RELATED parameter may be used, for example, as follows:

```
LOAD1   LOAD       EP=APGIOHK1,RELATED=(DEL1,'LOAD APGIOHK1')
  .
  .
  .
DEL1    DELETE     EP=APGIOHK1,RELATED=(LOAD1,'DELETE APGIOHK1')
```

Return and Reason Codes

When the LOAD macro returns control to the caller, GPR 15 is set to zero if the load request was successful. If the load request was not successful and a caller-provided error routine (specified using the ERRET keyword) receives control, GPR 1 contains the abend code for the abend that would have been issued had the caller not provided an ERRET exit. GPR 15 contains the reason code associated with the abend code in GPR 1.

Example 1

Bring a load module containing a specified entry name (PGMLKRUS) into virtual storage. Let the system find the module from available libraries.

```
LOAD     EP=PGMLKRUS
```

Example 2

Bring a load module containing the entry name EPNAME into virtual storage. Indicate that register 7 contains the address of the DCB associated with the partitioned data set that contains this load module. Return the load address of the requested module in the location pointed to by register 8. If an error occurs during this processing, transfer control to the error routine located at ERRADDR.

```
LOAD EP=EPNAME,DCB=(7),LOADPT=(8),ERRET=ERRADDR
```

LOAD—List Form

The list form of the LOAD macro builds a nonexecutable problem program parameter list that can be referred to or modified by the execute form of the LOAD macro.

Syntax

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

---

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

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

EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address.
DE=*list entry addr*	*list entry addr*: A-type address.
,DCB=*dcb addr*	*dcb addr*: A-type address.
,LSEARCH=NO ,LSEARCH=YES	**Default**: No
,LOADPT=*addr*	*addr*: A-type address.
,EXTINFO=*addr*	*addr*: A-type address.
,RELATED=*value*	
,SF=L	

Parameters

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

,SF=L
Specifies the list form of the LOAD macro.

LOAD—Execute Form

The execute form of the LOAD macro can refer to and modify the parameter list constructed by the list form of the macro.

Syntax

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

---

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

EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: RX-type address, or register (2) - (12).
DE=*list entry addr*	*list entry addr*: RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: RX-type address, or register (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr*: RX-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: No
,LOADPT=*addr*	*addr*: RX-type address or register (2) - (12).
,EXTINFO=*addr*	*addr*: A-type address.
,RELATED=*value*	
,SF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12) or (15).

Parameters

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

,SF=(E,*list addr*)
Specifies the execute form of the LOAD macro.

LSEXPAND — Expand a Linkage Stack to a Specified Size

Description

The LSEXPAND macro expands a normal linkage stack or a recovery linkage stack to a specified number of entries. The work unit associated with the calling program uses the **normal linkage stack** to save program status information. When the system needs an entry and finds that all entries are used, it issues a “stack full” program interruption. After the “stack full” interruption occurs, the system uses the **recovery linkage stack** for recovery.

If a program does not specify the LSEXPAND macro, it receives a normal linkage stack with 96 entries and a recovery linkage stack with 24 entries.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	31-bit
ASC mode:	Primary or AR
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Not applicable

Programming Requirements

If the system has already issued a stack full program interruption, the system will not accept the LSEXPAND macro. In other words, do not wait until the normal or recovery linkage stacks are full to issue this macro.

Restrictions

None.

Input Register Information

Before issuing the LSEXPAND 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	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

© Copyright IBM Corp. 1988, 2002

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 LSEXPAND macro is written as follows:

---

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

NORMAL=*n*	*n*: Symbol or number or value in register (2) - (12).
RECOVERY=*n*	*n*: Symbol or number or value in register (2) - (12).

Parameters

LSEXPAND

Specifies the number of entries that a task has for its normal linkage stack or its recovery linkage stack.

NORMAL=*n*

Specifies the number of entries in the normal linkage table, where *n* can be between 97 and 16000. If you don't specify this parameter, the normal linkage stack has 96 entries.

RECOVERY=*n*

Specifies the number of entries in the recovery linkage stack, where *n* can be between 25 and 4000. If you don't specify this parameter, the recovery linkage stack has 24 entries.

ABEND Codes

None.

Return Codes

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

Table 25. Return and Reason Codes for the LSEXPAND Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Successful completion. **Action**: None.
08	**Meaning**: Program error. The caller was not unlocked. **Action**: Release locks before calling LSEXPAND.
0C	**Meaning**: Program error. The caller was not in task mode. **Action**: Change your code to run in task mode.
10	**Meaning**: Program error. The specified normal stack size exceeds 16000. **Action**: Specify a stack size less than 16000.
14	**Meaning**: Program error. The specified recovery stack size exceeds 4000. **Action**: Specify a stack size less than 4000.
18	**Meaning**: Program error. The recovery stack cannot be expanded because it is currently in use. **Action**: Restructure your program to issue the LSEXPAND before the stack becomes full.
1C	**Meaning**: Program error. The normal stack cannot expand because the specified value is smaller than the current normal stack size. **Action**: Specify a larger stack size.
20	**Meaning**: Program error. The recovery stack cannot expand because the specified value is smaller than the current recovery stack size. **Action**: Specify a larger stack size.
24	**Meaning**: Environmental error. Not enough virtual storage was available for the normal linkage stack or the recovery linkage stack. **Action**: Retry the request one or more times. If the problem persists, check with the operator to see why there is a storage constraint.
28	**Meaning**: System error. The normal linkage stack is unchanged. The recovery linkage stack might be expanded. **Action**: Retry the request.

Example 1

Expand the normal linkage stack to 192 entries.

```
LSEXPAND NORMAL=192
```

Example 2

Expand the recovery linkage stack to 96 entries.

```
LA   6,96
LSEXPAND RECOVERY=(6)
```

PGLOAD — Load Virtual Storage Areas into Central Storage

Description

Note: IBM recommends that you use the PGSER macro rather than PGLOAD.

The PGLOAD macro is used to load specified virtual storage areas into central (also called real) storage in anticipation of future needs. That is, PGLOAD is essentially a page-ahead function. The PGLOAD macro performs this function for virtual addresses below 16 megabytes; the LOAD option of the PGSER macro performs the same function for virtual addresses either above or below 16 megabytes. Note, however, that a page that has been loaded via PGLOAD is eligible for page-out selection in the same manner as a page that has been demand-paged into central storage.

The misuse of this function can have adverse effects on system performance. Causing unnecessary pages to be brought into central storage will force other pages to be displaced and, consequently, cause unnecessary paging activity. Proper use of this function, however, will tend to decrease system overhead resulting from page faults.

Syntax

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

---

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

R	
,A=*start addr*	*start addr:* A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr:* A-type address, or register (0) or (2) - (12).
,EA=*end addr*	*end addr:* A-type address, or register (2) - (12) or (15). **Default:** *start addr* + 1
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N **Note:** RELEASE=Y may only be specified with EA above.

Parameters

The parameters are explained as follows:

© Copyright IBM Corp. 1988, 2002

R Specifies that no parameter list is being supplied with this request.

,A=*start addr*
Specifies the start address of the virtual area to be loaded.

,ECB=*ecb addr*
Specifies the address of an ECB that is used to signal event completion.

,EA=*end addr*
Specifies the end address + 1 of the virtual area to be loaded.

,RELEASE=N
,RELEASE=Y
Specifies that the contents of the virtual area is to remain intact (N) or be released (Y).

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Operation completed normally; ECB posted complete.
08	Operation proceeding; ECB will be posted when all page-ins are complete.

If control is not returned, an ABEND is issued with the following reason codes in register 15:

Hexadecimal Code	Meaning
10	Virtual subarea list entry or ECB address invalid. No ECB is posted.

If the ECB parameter is coded, the ECB is unchanged if the request was initiated but not complete (return code 8), or if an ABEND was issued with return code 10. Otherwise, the ECB is posted complete with code

0 - Operation completed successfully.

If the return code issued is 8, the ECB is posted asynchronously when paging I/O has completed, with code

0 - Operation completed successfully.

Example 1

Page-in a single byte of virtual storage, causing the entire 4096-byte page containing that byte to be paged into central storage.

```
PGLOAD   R,A=(R3)
```

Example 2

Page-in the virtual storage lying in the range addressed by registers 3 and 4, and notify the requestor via posting of the ECB when the page-ins are complete.

```
PGLOAD   R,A=(R3),EA=(R4),ECB=(R5)
```

Example 3

Discard the contents of the virtual pages totally encompassed by STARTAD and ENDAD before new real frames are assigned.

```
PGLOAD   R,A=STANDARD,EA=ENDAD,RELEASE=Y
```

PGLOAD—List Form

The list form of the PGLOAD macro uses a virtual subarea list.

Syntax

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

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

L	
,LA=*list addr*	*list addr*: A-type address, or register (1) or (2) - (12).
,ECB=*ecb addr*	*ecb addr*: A-type address, or register (0) - (2) or (15).
,RELEASE=N ,RELEASE=Y	**Default:** RELEASE=N

Parameters

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

L Specifies that a parameter list is being supplied with this request.

,LA=*list addr*
Specifies the address of the first entry of a virtual subarea list.

PGOUT — Page Out Virtual Storage Areas from Central Storage

Description

Note: IBM recommends that you use the PGSER macro rather than PGOUT.

The PGOUT macro is used to initiate page-out operations for specified virtual storage areas that are in central (also called real) storage. The PGOUT macro performs this function for virtual addresses below 16 megabytes; the OUT option of the PGSER macro performs the same function for virtual addresses either above or below 16 megabytes. The PGOUT function is complementary to the PGLOAD function. You have the option of specifying that the virtual pages to be paged out either remain valid in central storage, or be marked invalid and the real frames assigned to them be made available for reuse. The use of this option will not prevent page faults from occurring on the specified storage.

The misuse of this function, like the misuse of the PGLOAD function, can have adverse effects on system performance. On the other hand, proper use of this function will tend to clean out of central storage those pages no longer needed for program execution or not required for some period in the future.

Syntax

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

---

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

R	
,A=*start addr*	*start addr*: A-type address, or register (1) or (2) - (12).
,EA=*end addr*	*end addr*: A-type address, or register (2) - (12) or (15).
,KEEPREL=N ,KEEPREL=Y	**Default**: KEEPREL=N

Parameters

The parameters are explained as follows:

R Specifies that no parameter list is being supplied with this request.

© Copyright IBM Corp. 1988, 2002

,A=*start addr*
Specifies the start address of the virtual area to be paged out.

,EA=*end addr*
Specifies the end address + 1 of the virtual area to be paged out.

,KEEPREL=N
,KEEPREL=Y
Specifies that the virtual pages will be marked invalid and the real frames freed for reuse (N) or that the virtual pages will not be invalidated (Y).

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Operation completed normally; paging I/O proceeding asynchronously.
0C	One or more pages specified to be paged out were not paged out. Either the pages were in the nucleus in unusable real frames, in SQA or LSQA, in V=R area allocated region, were page fixed, or the system resources necessary to perform the page out operations were momentarily unavailable. Paging I/O is proceeding normally for all other pages.
10	Operation abnormally terminated. Virtual subarea list entry invalid.

Example 1

Page out the area of central storage totally encompassed by the start and end virtual boundaries specified.

```
PGOUT   R,A=(R3),EA=(R4)
```

Example 2

Create an auxiliary storage copy of a virtual area before continuing to use the area. The area will remain in central storage after the page-outs complete.

```
PGOUT   R,A=(R3),EA=(R4),KEEPREL=Y
```

PGOUT—List Form

The list form of the PGOUT macro uses a virtual subarea list.

Syntax

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

---

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

L	
,LA=*list addr*	*list addr*: A-type address, or register (1) or (2) - (12).
,KEEPREL=N ,KEEPREL=Y	**Default**: KEEPREL=N

Parameters

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

L Specifies that a parameter list is being supplied with this request.

,LA=*list addr*
Specifies the address of the first entry of a virtual subarea list (VSL). See the topic “Virtual Subarea List (VSL)” in *z/OS MVS Programming: Assembler Services Guide* for a description of the VSL.

PGRLSE — Release Virtual Storage Contents

Description

Note: IBM recommends that you use the PGSER macro rather than PGRLSE.

The PGRLSE macro is used to release to the system all central (also called real) storage and auxiliary storage associated with specified pageable virtual storage areas. The PGRLSE macro performs this function for virtual addresses below 16 megabytes; the RELEASE option of the PGSER macro performs the same function for virtual addresses either above or below 16 megabytes. Use PGRLSE when a large area (one or more complete pages) of virtual storage within your program no longer has significant contents.

Functionally, PGRLSE is equivalent to a FREEMAIN macro followed by a GETMAIN macro. That is, the virtual space is maintained, but the data is discarded. When a released page is next referred to, its contents are binary zeros. Thus, you can help reduce system overhead by releasing virtual storage when you no longer need it.

Proper use of this function can increase the amount of storage available to the system and prevent needless paging I/O activity. Usage of PGRLSE may improve operating efficiency when the using program can discard the contents of a large virtual storage area and reuse the virtual storage pages; paging operations may be eliminated for those virtual storage pages when they are reused.

Syntax

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

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

LA=*low addr*	*low addr*: A-type address, or register (0) or (2) - (12).
,HA=*high addr*	*high addr*: A-type address, or register (1) or (2) - (12).

Parameters

The parameters are explained as follows:

LA=*low addr*
Specifies the address of the lower boundary of the area to be released.

© Copyright IBM Corp. 1988, 2002

,HA=*high addr*
Specifies the address of the upper boundary + 1 of the area to be released.

When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion.
04	Execution failed. The area specified, or a portion of the area, is protected from the requesting program. Any valid portion of the area preceding the protected area is released.

Example 1

Release the contents of the pages included within the specified areas. Only those pages fully encompassed will be nullified.

```
PGRLSE   LA=(R4),HA=(R5)
```

Example 2

Perform the operation in Example 1, but use A-type addresses.

```
PGRLSE   LA=LOWADDR,HA=HIGHADDR
```

PGRLSE—List Form

The list form of the PGRLSE macro is used to construct a control program parameter list.

Syntax

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

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

LA=*low addr*,	*low addr*: A-type address.
,HA=*high addr*,	*high addr*: A-type address.
,MF=L	

Parameters

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

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

PGRLSE—Execute Form

A remote control program parameter list is referred to, and can be modified by, the execute form of the PGRLSE macro.

Syntax

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

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

LA=*low addr*,	*low addr*: A-type address, or register (0) or (2) - (12).
,HA=*high addr*,	*high addr*: A-type address, or register (1) or (2) - (12).
,MF=(E,*ctrl addr*)	*ctrl addr*: RX-type address, or register (2) - (12).

Parameters

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

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

PGSER — Page Services

Description

Note: IBM recommends that you use the PGSER macro for paging services.

The PGSER macro performs the same paging services as the PGLOAD, PGOUT, and PGRLSE macros. PGSER performs these services for addresses either above or below 16 megabytes.

The services are:

- Page load equivalent to the PGLOAD macro.
- Page out equivalent to the PGOUT macro.
- Page release equivalent to the PGRLSE macro.
- The PGSER macro with the PROTECT parameter makes a range of virtual storage pages read-only.
- The PGSER macro with the UNPROTECT parameter makes a range of virtual storage pages modifiable.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state, and any PSW key. To use the PROTECT and UNPROTECT options, the caller must have a PSW key that matches the key of the storage.
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 primary address space

Programming Requirements

- The caller must include the IHAPVT mapping macro.
- Regardless of the addressing mode, all addresses passed in registers are used as 31-bit addresses.
- All RX-type addresses are assumed to be in the addressing mode of the caller.

Restrictions

None.

Input Register Information

Before issuing the PGSER 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:

© Copyright IBM Corp. 1988, 2002

Register	Contents
0-4	Used as work registers by the system
5-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the access registers (ARs) are 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 PGSER macro is written as follows:

---

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

R
L

,LOAD
,OUT
,PROTECT
,UNPROTECT
,RELEASE

,LA=*list addr*	*list addr*: RX-type address or register (1), (2) - (12). **Note**: This parameter is valid only with L.
,A=*start addr*	*start addr*: RX-type address or register (1), (2) - (12). **Note**: This parameter is valid only with R.
,EA=*end addr*	**Default**: EA=*start addr* *end addr*: RX-type address or register (15), (2) - (12). **Note**: This parameter is valid only with R.
,ECB=*ecb addr*	**Default**: If LOAD is specified, ECB=0. *ecb addr*: RX-type address or register (0) or (2) - (12). **Note**: This parameter is optional if LOAD is specified and is not valid for OUT and RELEASE.

,RELEASE=Y ,RELEASE=N	**Default**: RELEASE=N **Note**: This parameter may be specified only if LOAD is specified.
,KEEPREL=Y ,KEEPREL=N	**Default**: KEEPREL=N **Note**: This parameter may be specified only if OUT is specified.
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

R
L Specifies the manner in which the input is supplied. If R is specified, the user supplies the starting and ending addresses of the virtual area for which the service needs to be performed. If L is specified, the user supplies the address of the page services list (PSL), which specifies the virtual area for which the service is to be performed. See the topic “Page Service List (PSL)” in *z/OS MVS Programming: Assembler Services Guide* for a description of the PSL.

,LOAD
,OUT
,PROTECT
,UNPROTECT
,RELEASE
Indicates the function to be performed.

LOAD specifies that a page-in operation is to be initiated for the virtual storage area specified, in anticipation of future needs.

OUT specifies that a page-out operation is to be initiated for the virtual storage area specified.

PROTECT specifies that a range of virtual storage be made read-only. R, L, LA, A, EA, and RELATED are valid keywords with the PROTECT option.

UNPROTECT specifies that a range of virtual storage be made modifiable. R, L, LA, A, EA, and RELATED are valid keywords with the UNPROTECT option.

RELEASE specifies the release of all physical paging resources, including both processor storage and auxiliary storage. Functionally, RELEASE is equivalent to a FREEMAIN macro followed by a GETMAIN macro. That is, the virtual space is maintained, but the data is discarded. When a released page is next referred to, its contents are binary zeros.

Note: You must unprotect protected storage before releasing it.

,LA=*list addr*
Specifies the address of the page services list (PSL) for L requests.

,A=*start addr*
Specifies the address of the start of the virtual area for R requests.

,EA=*end addr*
Specifies the address of the last byte on the last page of the virtual area for R requests.

,ECB=*ecb addr*
Specifies the address of the ECB that is used to signal event completion for a LOAD request.

If an ECB is supplied, the caller must check the return code because the ECB will not be posted if the return code is zero. If an ECB is not supplied, it is not necessary to check the return code because control returns to the caller only if the request was successfully completed; if unsuccessful, page services abnormally terminates the caller. You must ensure that the storage area containing the ECB is not freed and that the key is not altered. If either test fails, page services does not post the ECB.

,RELEASE=Y
,RELEASE=N
Specifies that all the central (also called real) and auxiliary storage associated with the virtual storage areas is to be released to the system (Y), or that all the central and auxiliary storage associated with the virtual storage areas is not to be released to the system (N).

,KEEPREL=Y
,KEEPREL=N
Specifies that the virtual pages should be validated again after the page-out completes (Y), or that the virtual pages will be marked invalid and the real frames freed for reuse (N).

,RELATED=*value*
Provides information to document the macro by relating the service performed to some corresponding function or service. The format can be any valid coding value that the user chooses.

ABEND Codes

PGSER might abnormally terminate with one of the following abend codes: X'18A', X'28A'. See *z/OS MVS System Codes* for explanations and programmer responses.

Return and Reason Codes

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

Option	Code	Meaning and Action
LOAD	0	**Meaning**: The operation completed normally and the ECB will not be posted. If no ECB is supplied, the operation is completed or proceeding. **Action**: None. If the ECB parameter was specified, do not issue a WAIT macro for the ECB after receiving this return code because it will not be posted.
LOAD	8	**Meaning**: The operation is proceeding. The ECB, if applicable and available, will be posted with X‘00’ when all page-ins are complete. **Action**: None. However, if the ECB parameter was specified, issuing a WAIT macro for this ECB will allow your program to synchronize with the completion of the page load operation.
OUT	0	**Meaning**: The operation completed normally. **Action**: None.

Option	Code	Meaning and Action
OUT	C	**Meaning**: At least one page specified to be paged out was not paged out. The page service is proceeding for the other pages. **Action**: None.
RELEASE	0	**Meaning**: The operation completed normally. **Action**: None.

Examples

Example 1

Perform the page-load function for the 4096-byte virtual area starting at BUFFER, supplying no ECB. Include the IHAPVT mapping macro.

```
PGSER R,LOAD,A=BUFFER,EA=BUFFER+4095,ECB=0
IHAPVT
```

Example 2

Release the virtual area specified in the PSL located at LOADWORD. Include the IHAPVT mapping macro.

```
PGSER L,RELEASE,LA=LOADWORD
IHAPVT
```

Example 3

Protect the storage area that starts at the address in GPR 4 and ends at the address in the variable ENDIT. Include the IHAPVT mapping macro.

```
PGSER R,PROTECT,A=(4),EA=ENDIT
IHAPVT
```

POST — Signal Event Completion

Description

Use the POST macro to set an event control block (ECB) to indicate the occurrence of an event. If this event satisfies the requirements of an outstanding WAIT or EVENTS macro, the waiting task is taken out of the wait state and dispatched according to its priority. POST processing sets the bits in the ECB as follows:

Bit 0 to 0 (wait bit)
Bit 1 to 1 (complete bit)
Bits 2 through 31 to the specified completion code.

Note: After the bits in the ECB are set, the ECB is considered posted and the awaited event can be recognized as having occurred by programs running in the system. If a program issues another POST against an ECB that is already posted, the other POST has no effect.

For more information on how to use the POST macro to synchronize tasks, see *z/OS MVS Programming: Assembler Services Guide*.

Environment

The requirements for callers of POST are:

Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	One of the following: • 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:	• For LINKAGE=SVC: No locks held and no enabled unlocked task (EUT) functional recovery routines (FRR) established • For LINKAGE=SYSTEM: No locks held
Control parameters:	The event control block (ECB) must be in the primary address space.

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the POST 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 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 LINKAGE=SVC is specified: Used as a work register by the system • If LINKAGE=SYSTEM is specified: Return code of 0

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 POST macro is written as follows:

---

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

---

ecb addr	*ecb addr*: RX-type address, or register (1) or (2) - (12).
,*comp code*	*comp code*: Symbol, decimal digit, or register (0) or (2) - (12). **Range of values**: 0 to 2^{30}-1 **Default**: 0
,LINKAGE=SVC ,LINKAGE=SYSTEM	**Default**: LINKAGE=SVC
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The explanation of the parameters is as follows:

ecb addr
Specifies the address of the fullword event control block representing the event.

,comp code
Specifies the completion code to be placed in the event control block upon completion.

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

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. The ECB must be in the caller's primary address space. LINKAGE=SYSTEM is intended to be used by programs in cross memory mode.

The default is LINKAGE=SVC.

,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.

The RELATED parameter is available on macros that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE) and on macros that relate to previous occurrences of the same macros (for example, CHAP and ESTAE).

The RELATED parameter may be used, for example, as follows:

```
WAIT1   WAIT    1,ECB=ECB,RELATED=(RESUME1,'WAIT FOR EVENT')
  .
  .
  .
RESUME1 POST    ECB,0,RELATED=(WAIT1,'RESUME WAITER')
```

Return and Reason Codes

For LINKAGE=SYSTEM, the return code in register 15 is always zero. Otherwise, the POST macro has no return codes.

Example 1

Signal event completion with a default completion code. POSTECB is the address of an ECB.

```
POST     POSTECB
```

Example 2

Signal event completion with a completion code of X'7FF'. POSTECB is the address of an ECB.

```
POST     POSTECB,X'7FF'
```

QRYLANG — Determine Languages Available for Message Translation

Description

The QRYLANG macro enables you to check if a particular language is available into which you can translate system or application messages. It can also provide a list of all active languages currently available for translation. Once you know that the language you want is available, you can issue TRANMSG to retrieve the translated message.

QRYLANG returns the information you request in the language query block (LQB). This block contains the following:

- The standard 3-character code for the language
- The name of the language
- A flag indicating whether the language contains double-byte characters

If you asked for a list of all available languages, QRYLANG returns an LQB with one language entry for each language.

See *z/OS MVS Programming: Assembler Services Guide* for more information on using QRYLANG.

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 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:	Not applicable

Programming Requirements

Before invoking QRYLANG you must allocate storage for the LQB.

You must include the following mapping macros:

- CNLMLQB
- CNLMMCA

Restrictions

None.

Input Register Information

Before issuing the QRYLANG macro, the caller must ensure that the following general purpose register (GPR) contains the specified information:

Register	Contents
13	Points to a save area

© Copyright IBM Corp. 1988, 2002

Output Register Information

When control returns to the caller, the output registers contain:

Register	Contents
0	• The contents of the high-order halfword are not part of the intended programming interface. • The low-order halfword contains a reason code.
1	Used as a work register by system
2-13	Unchanged
14	Used as a work register by system
15	Return code

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

Performance Implications

None.

Syntax

The QRYLANG macro is written as follows:

---

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

LQB=*lang qblock addr*	*lang qblock addr*: RX-type address or register (2) - (12).
,LQBLEN=*length of block addr*	*length of block addr*: RX-type address or register (2) - (12).
,LANGNAME=*lang addr*	*lang addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

LQB=*lang qblock addr*
Specifies the storage area or a register pointing to the storage area where QRYLANG is to build the LQB.

,LQBLEN=*length of block addr*
Specifies the fullword or a register containing the length in bytes of the LQB.

You must supply the length of the LQB if you are querying more than one language. See *z/OS MVS Programming: Authorized Assembler Services Guide* for information on how to calculate the length of the LQB. If you do not specify LQBLEN, QRYLANG will default to the assembled length of the LQB parameter. If you use an RX-type address or register notation for the LQB parameter, you must specify LQBLEN.

,LANGNAME=*lang addr*
Specifies the 24-byte character field or a register pointing to the 24-byte character field containing the name or code of the language to be queried. See *z/OS MVS Programming: Assembler Services Guide* for a listing of the language codes. The language name must match the name specified on the NAME parameter of the LANGUAGE statement in the MMSLSTxx member of SYS1.PARMLIB. If you omit this keyword, QRYLANG returns a list of all currently available languages.

Return and Reason Codes

When QRYLANG completes, register 15 contains one of the following hexadecimal return codes:

Hexadecimal Code	Meaning
00	Processing completed successfully.
04	Processing did not complete, and storage is not freed.
08	Processing is complete but QRYLANG returned an incomplete LQB to the calling program. For example, the requested language may not be available.
0C	Processing did not complete. The output is unusable.
10	The function did not complete. The output LQB is unusable.

The low-order halfword of register 0 contains the following hexadecimal reason codes from QRYLANG:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00	00	Successful processing.
04	07	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	08	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	0B	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	0C	The passed storage address is not valid.
04	0D	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
08	0F	There is insufficient LQB storage for LQB entries.
08	2C	The language you requested is not available.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
0C	0A	No storage was obtained.
0C	16	The LQB is too small to handle returned data.
0C	17	The MVS message service is not available.
0C	26	The query request terminated. The MMS user exit has set the processing indicator to a nonzero value.
0C	27	The entry installation exit has failed.
0C	28	The exit installation exit has failed.
0C	2D	The acronym of the control block created when invoking QRYLANG is not ″LQB″ and is therefore not valid.
0C	2E	The length of the LQB is not valid.
0C	2F	QRYLANG was unable to move the LQB from the caller's address space.
0C	30	QRYLANG was unable to move the LQB to the caller's address space.
10	09	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

Example

Check if the language with a language code of JPN is active. If JPN is active, QUERY2A sets a flag within the installation-created control block to ″on″, indicating that JPN is available.

```
QUERY2A  CSECT
QUERY2A  AMODE 31
QUERY2A  RMODE ANY
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
         ST    13,SAVE+4
         LA    15,SAVE
         ST    15,8(13)
         LR    13,15
*                                                                     *
***********************************************************************
*        OBTAIN STORAGE AREA FOR INSTLCB AND LQB                      *
***********************************************************************
*                                                                     *
         GETMAIN RU,LV=STORLEN,SP=SP228
*                                                                     *
         LR    R3,R1                SAVE ADDRESS OF STORAGE AREA
         ST    R3,CVTUSER-CVT(R2)   ANCHOR INSTALLATION CONTROL BLOCK  C
                                    FROM GLOBAL COMMUNICATIONS WORD    C
                                    IN MCA CONTROL BLOCK
         XC    0(STORLEN,3),0(3)    CLEAR STORAGE AREA
         MVC   INSTLACR-INSTLCB(4,R3),=C'INST'  SET ACRONYM IN         C
                                          INSTALLATION CONTROL BLOCK
         LA    R4,INSTLLEN(,R3)     OBTAIN ADDRESS OF LQB
         LA    R5,LQBLEN            GET LQB LENGTH
*                                                                     *
         QRYLANG LANGNAME=JPN_CODE,LQB=(R4),LQBLEN=(R5)
*                                                                     *
         LTR   R15,R15              IS JAPANESE AVAILABLE
         BNZ   END                  NO, EXIT
```

```
         OI    INSTLFLG-INSTLCB(R3),INSTLJPN    YES, SET AVAIL. FLAG
*
*********************************************************************
*        RETURN                                                     *
*********************************************************************
*
END      DS    0H
         L     13,SAVE+4
         LM    14,12,12(13)
         BR    14
*********************************************************************
JPN_CODE DC    CL24'JPN'
SAVE     DC    18F'0'
SP228    EQU   228
LQBLEN   EQU   (LQBVDAT-LQB)+LQBEBL
STORLEN  EQU   INSTLLEN+LQBLEN
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R15      EQU   15
*********************************************************************
         DSECT
         CVT   DSECT=YES
         CNLMMCA
         CNLMLQB
INSTLCB  DSECT               INSTALLATION CONTROL BLOCK
INSTLACR DS    CL4'INST'     INSTALLATION CONTROL BLOCK ACRONYM
INSTLFLG DS    X             LANGUAGE AVAILABILITY FLAGS
INSTLJPN EQU   X'80'         JAPANESE IS AVAILABLE
         DS    CL23          RESERVED
INSTLLEN EQU   *-INSTLCB
         END QUERY2A
```

REFPAT — Define and End a Reference Pattern

Description

The REFPAT macro identifies a large data area and tells the system how the program will be referencing that area. Additionally, the program tells the system how many bytes of data it wants the system to bring into central storage on a page fault (that is, each time the program references data that is not in central storage). Use REFPAT if your program accesses a very large data area in a reference pattern that is consistently in a forward or backward direction. The system responds to REFPAT by bringing multiple pages into central storage on a page fault. REFPAT might significantly improve the performance of the program.

REFPAT INSTALL defines the reference pattern and REFPAT REMOVE removes the definition.

Your program can reference an area with one pattern, then later reference the same area with another pattern. Use REFPAT INSTALL to define the first reference pattern and REFPAT REMOVE to remove the definition. Then, issue REFPAT INSTALL to define another pattern for the same area.

On REFPAT INSTALL, you describe the data area, the reference pattern, and tell the system how many bytes of data you want it to bring into central storage on a page fault. Two parameters, UNITSIZE and GAP, determine the reference pattern:

- UNITSIZE specifies the size of a “reference unit”. A reference unit is a grouping of contiguous bytes that the program references. You might decide a reference unit is the group of bytes that make up an element of an array, or the group of bytes that occur between gaps, or a page (4096 bytes).
- GAP defines the size of “gaps” in the reference pattern. Gaps are areas that the program does not reference; they must be uniform in size and appear throughout the data area at repeating intervals. Not all reference patterns include such a gap.

UNITS specifies how many reference units, as defined on UNITSIZE, you want the system to bring into central storage on a page fault.

The data area can be located in the primary address space, or in a data space identified by the STOKEN parameter.

Each pattern defined by REFPAT INSTALL is associated with the task that represents the caller. A task can have up to 100 reference patterns for different data areas, but cannot have multiple patterns for the same area. Multiple tasks can specify a different reference pattern for the same data area. REFPAT REMOVE removes the association between the pattern and the task.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts

© Copyright IBM Corp. 1988, 2002

Locks: No locks held
Control parameters: Must be in the primary address space.

Programming Requirements

If your program is in AR mode, make sure the SYSSTATE ASCENV=AR macro has been issued to tell the system to generate code appropriate for AR mode.

Restrictions

If you specify STOKEN for a data space, the data space must be owned by a task in the primary address space.

Input Register Information

Before issuing the REFPAT 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 the return code in GPR 15 is not 0; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

When control returns to the caller, the ARs contain:

Register	Contents
1	Used as a work register 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

The system rejects the REFPAT macro if the values you specify do not benefit the performance of your program. To make sure the system accepts the macro, ask the system to bring in more than three pages (that is, 12288 bytes) on each page fault.

Syntax

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

---

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

b — One or more blanks must precede REFPAT.

REFPAT	
b	One or more blanks must follow REFPAT.

INSTALL REMOVE	
,PSTART=*start* ,PEND=*end*	*start*: RX-type address or address in register (2) - (12). *end*: RX-type address or address in register (2) - (12).
,STOKEN=*stoken*	*stoken*: RX-type address or register (2) - (12). **Default**: STOKEN=0
,UNITSIZE=*unit size*	*unit size*: RX-type address or register (2) - (12). UNITSIZE is required with INSTALL.
,GAP=*gap variable*	*gap variable*: RX-type address or register (2) - (12). **Default**: GAP=0
,UNITS=*unit number*	*unit number*: RX-type address or register (2) - (12). **Default**: UNITS=1

Parameters

The parameters are explained as follows:

INSTALL
REMOVE

INSTALL indicates that the program is to begin referencing the data area according to a defined pattern. Required parameters on the INSTALL request are PSTART, PEND, and UNITSIZE. UNITS, GAP, and STOKEN are optional.

REMOVE indicates that the program has finished referencing the data area, as specified by the previous REFPAT INSTALL request. Required parameters on the REMOVE request are PSTART and PEND. STOKEN is optional on the REMOVE request; UNITSIZE, GAP, and UNITS are not valid.

PSTART and PEND on the INSTALL request must be exactly the same as PSTART and PEND on the REMOVE request for the same reference pattern.

,PSTART=*start*

A required parameter that contains the address of the first byte of the data area for which the reference pattern applies. PSTART and PEND addresses must not straddle the common area boundaries. That is, for data in the primary address space, all data must be either in low private, in common, or in high private storage.

When a gap exists, define PSTART according to the following rules:

- If direction is forward, PSTART must be the first byte (low-address end) of a reference unit.

- If direction is backward, PSTART must be the last byte (high-address end) of a reference unit.

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

,PEND=*end*
A required parameter that contains the address of the last byte of the data area for which the reference pattern applies. If *start* is a higher address than *end*, the system knows that data reference is in a backward direction.

Whether or not a gap exists, PEND can be any part of a reference unit or a gap.

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

,STOKEN=*stoken*
Specifies the STOKEN that identifies the data space that contains the data area. You received the STOKEN either from DSPSERV or from another program.

If you use STOKEN=0 or do not specify STOKEN, the system assumes the data is in the primary address space.

,UNITSIZE=*unit size*
Specifies the number of consecutive bytes that you want the system to treat as a reference unit. If the pattern includes a gap, the reference unit is the grouping of bytes that lie between the gaps. If the pattern does not include a gap, you can use any logical grouping of bytes that your data structure suggests, such as an element, a row or two, or a page (4096 bytes). UNITSIZE is required for the INSTALL request.

,GAP=*gap variable*
Specifies the gap, in bytes, of the reference pattern. The default is GAP=0.

,UNITS=*unit number*
Specifies the number of reference units, as defined on UNITSIZE, the system is to page in at one time. The default is one reference unit or UNITS=1. To figure out how many bytes the system brings in at a time:

- If there is no gap, multiply the UNITS value by the UNITSIZE value and round up to the nearest 4096-byte boundary.
- If there is a gap, the number depends on values of UNITSIZE, GAP, UNITS, plus the location of the reference units and gaps relative to a page boundary. The system brings in the pages that contain the reference units. It does not bring in pages that contain only data in the gap. *z/OS MVS Programming: Assembler Services Guide* can help you code the parameters.

Return and Reason Codes

Return and reason codes, in hexadecimal, from REFPAT are:

Return Code	Reason Code	Meaning
00	None.	REFPAT completed successfully.
04	xx0001xx	REFPAT completed successfully; however, the system did not accept the reference pattern the caller specified. The system decided that the normal paging algorithms would be more efficient.

Return Code	Reason Code	Meaning
08	xx0002xx	Unsuccessful completion. The range that the caller specified on the INSTALL request overlaps the range that a previous request specified.
08	xx0003xx	Unsuccessful completion. The number of existing REFPAT INSTALL requests for the task exceeds 100, the maximum number the system allows.
08	xx0004xx	Unsuccessful completion. LSQA storage is not available for the macro service.
08	xx0101xx	Unsuccessful completion. The caller specified the REMOVE request; however, no INSTALL request was in effect for the specified range. Check to see if the system rejected the previous INSTALL request for the range.

Example 1

Define a reference pattern in which the program processes 8192 bytes and skips over 4096 bytes in a continuing way throughout an array. Registers 4 and 5 contain pointers to locations in storage which contain the starting and ending addresses of the array. Ask the system to bring in eight pages on each page fault.

```
REFPAT  INSTALL,PSTART=(4),PEND=(R5),GAP=4096,UNITSIZE=8192,UNITS=4
```

Example 2

Tell the system you have finished using the array using that pattern:

```
REFPAT  REMOVE,PSTART=(4),PEND=(R5)
```

REFPAT—List Form

Use the list form of the REFPAT macro together with the execute form of the macro for programs 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 REFPAT macro is written as follows:

---

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

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

Parameters

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

MF=(L,*list addr,attr***)**
Specifies the list form of the REFPAT macro. *list addr* defines the area that the system is to use 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.

REFPAT—Execute Form

Use the execute form of the REFPAT macro together with the list form of the macro for programs 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 REFPAT macro is written as follows:

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

INSTALL REMOVE	
,PSTART=*start*	*start*: RX-type address or register (2) - (12).
,PEND=*end*	*end*: RX-type address or register (2) - (12).
,STOKEN=*stoken*	*stoken*: RX-type address or register (2) - (12). **Default**: STOKEN=0
,UNITSIZE=*unit size*	*unit size*: RX-type address or register (2) - (12). UNITSIZE is required on INSTALL./.,pend
,GAP=*gap variable*	*gap variable*: RX-type address or register (2) - (12). **Default**: GAP=0
,UNITS=*unit number*	*unit number*: RX-type address or register (2) - (12).

Default: UNITS=1

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

Parameters

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

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the REFPAT macro. *list addr* defines the area that the system uses for the parameter list.

COMPLETE specifies that the system is to check for required parameters and supply optional parameters that are not specified.

RESERVE — Reserve a Device (Shared DASD)

Description

The RESERVE macro reserves a device for use by a particular system; it must be issued by each task needing to reserve a device shared with one or more systems. The RESERVE macro protects the caller from interference by other tasks in the system and locks out other systems. The reserve actually occurs when the first I/O is done to the device after the RESERVE macro is issued. When the reserving program no longer needs the reserved device, it should issue a DEQ macro to release the resource.

For information about how to obtain the UCB address for a device, see the section "Accessing Unit Control Blocks (UCBs)" in *z/OS MVS Programming: Assembler Services Guide* for information about using the UCBSCAN macro.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state with any PSW key.
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:	If the caller's AMODE is 24-bit, all parameters must reside below 16 megabytes.

Programming Requirements

None.

Restrictions

If a task issues two RESERVE macros for the same device without an intervening DEQ macro, an abnormal termination results unless the second RESERVE specifies the keyword parameter RET. (If a restart occurs after the caller successfully issued the RESERVE macro for a resource, the system does not reserve the device again; the caller must reissue the RESERVE macro.) If a DEQ macro is not issued for a particular resource, the system releases the reserved resource when the task ends.

The system counts and limits the number of concurrent resource requests in an address space. If an unconditional RESERVE (a RESERVE macro with RET=NONE) causes the number of global resource serialization requests to exceed the limit, the caller is abnormally terminated with a system code of X'538'. For further information about limiting concurrent requests for resources, see in *z/OS MVS Programming: Assembler Services Guide*.

© Copyright IBM Corp. 1988, 2002

Input Register Information

Before issuing the RESERVE 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	One of the following: • If you specify RET=TEST, RET=USE, or RET=HAVE: If all return codes for the resources named in the RESERVE 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 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 RESERVE macro is written as follows:

---

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

---

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

,	**Default:** E
,E	
,S	
,	
,rname length	*rname length*: symbol, decimal digit, or register (2) - (12).
,SYSTEMS	
)	
,RET=TEST	
,RET=USE	
,RET=HAVE	
,RET=NONE	
,UCB=*ucb addr*	*ucb addr*: A-type address, or register (2) - (12).
,LOC=BELOW	**Default:** LOC=BELOW
,LOC=ANY	
,RELATED=*value*	*value*: any valid macro keyword specification.
,LINKAGE=SVC	**DEFAULT**: LINKAGE=SVC
,LINKAGE=SYSTEM	

Parameters

The parameters are explained as follows:

(Specifies the beginning of the resource description.

qname addr
Specifies the address in virtual storage of an 8-character name. The name should not start with SYS, so that it will not conflict with system names. Every task issuing RESERVE against the same resource must use the same qname and rname to represent the resource.

,rname addr
Specifies the address in virtual storage of the name used together with qname to represent a single resource. The name can be qualified, and must be from 1 to 255 bytes long.

,
,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*.

,SYSTEMS
Specifies that the resource is shared among systems.

) Specifies the end of the resource description.

,RET=TEST
,RET=USE
,RET=HAVE
,RET=NONE
RET=TEST, RET=USE, and RET=HAVE specify a conditional request for the resource named on the macro, as follows:

RET=TEST The availability of the resource is to be tested, but control of the resource is not requested.

RET=USE Control of the resource is to be assigned to the active task only if the resource is immediately available.

RET=HAVE Control of the resource is requested only if the same task does not already control or have an outstanding request for the same resource.

RET=NONE specifies an unconditional request for the resource named on the macro.

,UCB=*ucb addr*
Specifies the address of a fullword that contains the address of the UCB for the device to be reserved. The UCB must be allocated to the job step before RESERVE is issued.

Note: The UCB keyword might specify a UCB address for a UCB that resides in storage above or below 16 megabytes. If the UCB address might point to a UCB above 16 megabytes you must also specify LOC=ANY.

,LOC=BELOW
,LOC=ANY
Specifies the location of the input UCB address. ANY specifies that the input UCB address is to be treated as a 31-bit address. BELOW specifies that the input UCB address is to be treated as a 24-bit address. The default is LOC=BELOW.

,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 values.

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

The default is LINKAGE=SVC.

ABEND Codes

For unconditional requests only, 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 return codes only if you specify RET=TEST, RET=USE, or RET=HAVE; for RET=NONE, return to the task indicates that control of the resource has been assigned to the task. If the return code for the resource named in the RESERVE macro is 0, register 15 contains 0. If the return code is not 0, register 15 contains the address of a 12-byte storage area containing the return code, as shown in Figure 4.

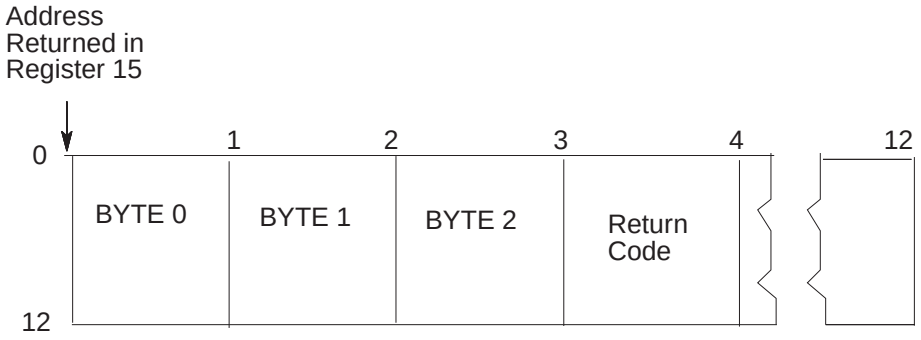

Figure 4. Return Code Area Used by RESERVE

The return codes for the RESERVE macro with the RET=TEST parameter are described in Table 26.

Table 26. Return Codes for the RESERVE 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.

Table 26. Return Codes for the RESERVE Macro with the RET=TEST Parameter (continued)

Hexadecimal Return Code	Meaning and Action
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 Byte 0 as shown in Figure 4 on page 509. 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.

The return codes for the RESERVE macro with the RET=USE parameter are described in Table 27.

Table 27. Return Codes for the RESERVE 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 Byte 0 as shown in Figure 4 on page 509. 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 27. Return Codes for the RESERVE 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 RESERVE macro with the RET=HAVE parameter are described in Table 28.

Table 28. Return Codes for the RESERVE Macro with the RET=HAVE Parameter

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: The active task now has control of the resource. **Action**: None.
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: None required. However, you might take some action based on your application. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of Byte 0 as shown in Figure 4 on page 509. 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.

Example

Unconditionally reserve exclusive control of a device. The length of the rname is allowed to default.

```
RESERVE (MAJOR3,MINOR3,E,,SYSTEMS),UCB=(R3)
```

RESERVE—List Form

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

---

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

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

(	
qname addr	*qname addr*: A-type address.
, ,*rname addr*	*rname addr*: A-type address.
, ,E ,S	
, ,*rname length*	*rname length*: symbol or decimal digit.
, ,SYSTEMS	
)	
,RET=TEST ,RET=USE ,RET=HAVE ,RET=NONE	
,UCB=*ucb addr*	*ucb addr*: A-type address or 0.
,LOC=BELOW ,LOC=ANY	**Default:** LOC=BELOW
,RELATED=*value*	*value*: A-type address.
,MF=L	

Parameters

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

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

RESERVE—Execute Form

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

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

(	**Note:** (and) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired, the (,), and all parameters between (and) should not be specified. If something in the list is desired, then (,), 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). **Note:** *rname length* must be coded if a register is specified for *rname addr* above.
, ,SYSTEMS	
)	
,RET=TEST ,RET=USE ,RET=HAVE ,RET=NONE	
,UCB=*ucb addr*	*ucb addr*: RX-type address, or register (2) - (12).
,LOC=BELOW ,LOC=ANY	**Default:** LOC=BELOW
,RELATED=*value*	*value*: any valid macro keyword specification.
,LINKAGE=SVC ,LINKAGE=SYSTEM	**DEFAULT**: LINKAGE=SVC
,MF=(E, *list addr*)	*list addr*: RX-type address, or register (1) - (12).

Parameters

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

,MF=(E,*ctrl addr)*

Specifies the execute form of the RESERVE macro.

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

RETURN — Return Control

Description

The RETURN macro restores the control to the calling program and signals normal termination of the called program. The return of control is always made by executing a branch instruction using the address in register 14. Because the RETURN macro uses a BR 14 to pass control, it can be used only when the return is to a program that executes in the same addressing mode. The RETURN macro can restore a designated range of registers, provide a return code in register 15, and flag the save area used by the called program.

If registers are to be restored, or if an indicator is to be placed into the save area, register 13 must contain the address of the save area, which must have the standard format.

Syntax

The RETURN macro is written as follows:

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

(*reg1*) (*reg1,reg2*)	*reg1* and *reg2*: Decimal digits, and in the order 14, 15, 0 through 12.
,T	
,RC=*ret code*	*ret code*: Decimal digit, symbol, or register (15). The maximum value is 4095.

Parameters

The parameters are explained as follows:

(reg1)
(reg1,reg2)
: Specifies the register or range of registers to be restored from the save area pointed to by the address in register 13. If you omit this parameter, the contents of the registers are not altered. Do not code this parameter when returning control from a program interruption exit routine.

,T Causes the control program to flag the save area used by the called program. The low-order bit of word 4 of the save area is set to 1 after the registers have

© Copyright IBM Corp. 1988, 2002

been loaded; this designates that a called program has executed a return to its caller. Do not specify this parameter when returning control from an exit routine.

,RC=*ret code*
Specifies the return code to be passed to the calling program. If a symbol or decimal digit is coded, the return code is placed right-adjusted in register 15 before return is made; if register 15 is coded, the return code has been previously loaded into register 15 and the contents of register 15 are not altered or restored from the save area. (If you omit this parameter, the contents of register 15 are determined by the *reg1* and *reg2* parameters.)

Note: If register 15 is coded and a return code greater than 4095 (decimal) is passed, the results could be either an invalid return code in the message or invalid RC testing.

Example

Restore registers 14-12, flag the save area, and return with a code of 0.

```
RETURN  (14,12),T,RC=0
```

SAVE — Save Register Contents

Description

The SAVE macro stores the contents of the specified general purpose registers in the save area at the address contained in register 13. If you wish, you may specify an entry point identifier. Write the SAVE macro only at the entry point of a program because the code resulting from the macro expansion requires that register 15 contain the address of the SAVE macro prior to its execution. Do not use the SAVE macro in a program interruption exit routine.

Syntax

The SAVE macro is written as follows:

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

(*reg1*) (*reg1,reg2*)	*reg1* and *reg2*: Decimal digits, and in the order 14, 15, 0 through 12.
, ,T	
,id name	*id name*: Character string of up to 70 characters or as an *.

Parameters

The parameters are explained as follows:

(*reg1***)**
(*reg1,reg2***)**
Specifies the register or range of registers to be stored in the save area at the address contained in register 13. The registers are stored in words 4 through 18 of the save area.

,
,T Specifies that registers 14 and 15 are to be stored in word 4 and 5, respectively, of the save area. This parameter permits you to save two noncontiguous sets of registers.

If you specify both T and *reg2*, and *reg1* is any of registers 14, 15, 0, 1, or 2, all of registers 14 through the *reg2* value are saved.

© Copyright IBM Corp. 1988, 2002

,id name
Specifies an identifier to be associated with the SAVE macro. If an asterisk (*) is coded, the identifier is the *name* associated with the SAVE macro, or, if the *name* field is blank, the control section name is used. The identifier aids in locating a program's save area in a dump. If the CSECT instruction name field is blank, the parameter is ignored.

Whenever a symbol or an asterisk is coded, the following macro expansion occurs:

- A count byte containing the number of characters in the identifier name is assembled four bytes following the address contained in register 15.
- The character string containing the identifier name is assembled starting at five bytes following the address contained in register 15.
- An instruction to branch around the count and identifier fields is assembled.

Example

Save registers 14-12, and associate the identifier with the CSECT name.

```
SAVE   (14,12),,*
```

SETRP — Set Return Parameters

Description

Use the SETRP macro within a recovery routine to indicate the various requests that the recovery routine can make. SETRP is valid for ESTAE-type recovery routines. For more information about recovery routines, see *z/OS MVS Programming: 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- or 64-bit
ASC mode:	Primary, secondary, or access register (AR) **Note:** Callers in secondary ASC mode cannot specify the DUMPOPX parameter.
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

- If the program is in AR mode, issue the SYSSTATE ASCENV=AR macro before issuing SETRP. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.
- Include the IHASDWA mapping macro to map the system diagnostic work area (SDWA). (See SDWA in *z/OS MVS Data Areas, Vol 4 (RD-SRRA)* for the mapping provided by IHASDWA.)
- If you plan to specify RETREGS=YES, RUB=*reg info addr*, you must obtain storage for and initialize the register update block (RUB). See the RETREGS parameter description for more information about this area.

Restrictions

- You can use SETRP only if the system provided an SDWA.
- Recovery routines established through the STAE macro, or the STAI parameter on the ATTACH or ATTACHX macro, cannot update registers on retry, so the RETREGS parameter does not apply.

Input Register Information

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

Register	Contents
1	If you do not specify the WKAREA parameter, address of the SDWA; otherwise, the caller does not have to place any information into this register.
13	If you specify the REGS parameter, address of a standard 72-byte save area containing the registers to be restored; otherwise, the caller does not have to place any information into this register.

© Copyright IBM Corp. 1988, 2002

Before issuing the SETRP macro, the caller must ensure that the following access registers (ARs) contain the specified information:

Register	Contents
1	If you do not specify the WKAREA parameter, ALET of the SDWA whose address is in GPR 1; otherwise, the caller does not have to place any information into this register.
13	If you specify the REGS parameter, ALET of the standard 72-byte save area whose address is in GPR 13; otherwise, the caller does not have to place any information into this register.

Output Register Information

Note: Control does not return to the caller if the caller specifies the REGS parameter.

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
14-15	Used as work registers 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 SETRP macro is written as follows:

---

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

,WKAREA=(*reg*)	*reg*: Decimal digits 1-12. **Default**: WKAREA=(1)

,REGS=(*reg1*) ,REGS=(*reg1,reg2*)	*reg1*: Decimal digits 0-12, 14, 15. *reg2*: Decimal digits 0-12, 14, 15. **Note**: If you specify *(reg1,reg2)*, specify the registers in the same order as in an STM instruction; for example, to restore all registers except register 13, specify REGS=(14,12).
,DUMP=IGNORE ,DUMP=YES ,DUMP=NO	**Default**: DUMP=IGNORE
,DUMPOPT=*parm list addr* ,DUMPOPX=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12). **Note**: Appropriate only with DUMP=YES.
,REASON=*code*	*code*: Any four-byte number specified in decimal (31-bit) or hexadecimal (32-bit).
,RC=0 ,RC=4 ,RC=16	**Default**: RC=0
,RETADDR=*retry addr*	*retry addr*: RX-type address, or register (2) - (12). **Note**: This parameter may be specified only if RC=4 is specified above.
,REMREC=NO ,REMREC=YES	**Default**: REMREC=NO
,RETREGS=NO ,RETREGS=YES ,RETREGS=YES,RUB=*reg info addr* ,RETREGS=64	*reg info addr*: RX-type address, or register (2) - (12). **Default**: RETREGS=NO **Note**: This parameter may be specified only if RC=4 is specified above.
,FRESDWA=NO ,FRESDWA=YES	**Default**: FRESDWA=NO **Note**: This parameter may be specified only if RC=4 is specified above.
,COMPCOD=*comp code* ,COMPCOD=(*comp code*,USER) ,COMPCOD=(*comp code*,SYSTEM)	*comp code*: Symbol, decimal digit, or register (2) - (12). **Default**: COMPCOD=(*comp code*,USER)
,RECPARM=*record list addr*	*record list addr*: RX=type address, or register (2) - (12).

Parameters

The parameters are explained as follows:

,WKAREA=(*reg***)**
Specifies the address of the SDWA passed to the recovery routine.

,REGS=(*reg1***)**
,REGS=(*reg1,reg2***)**
Specifies the register or range of registers to be restored from the 72-byte standard save area pointed to by the address in register 13. If you specify

REGS, a branch on register 14 instruction will also be generated to return control to the system. If you do not specify REGS, you must code your own branch on whichever register contains the return address.

Note: If you specify *reg1,reg2*, specify the registers in the same order as in an STM instruction; for example, to restore all registers except register 13, specify REGS=(14,12).

,DUMP=IGNORE
,DUMP=YES
,DUMP=NO

Specifies that the dump option fields will not be changed (IGNORE), will be zeroed (NO), or will be merged with dump options specified in previous dump requests, if any (YES). If IGNORE is specified, a previous recovery routine had requested a dump or a dump had been requested through the ABEND macro, and the previous request will remain intact. If NO is specified, no dump will be taken.

DUMP=YES does not guarantee that a SYSABEND/SYSUDUMP will be taken. You may specify this request in an FRR for an SRB but you will get an abdump only if the SRB abend successfully percolates to a task *and* none of the FRRs for that task choose to retry *and* the final value of the DUMP= remains the same after every recovery routine has received control.

,DUMPOPT=*parm list addr*
,DUMPOPX=*parm list addr*

Specifies the address of a parameter list of options. To create the parameter list, use the list form of either the SNAP or SNAPX macro, or code data constants in your program. DUMPOPT specifies the address of a parameter list that the SNAP macro creates. DUMPOPX specifies the address of a parameter list that the SNAPX macro creates. A program in secondary mode cannot use the DUMPOPX parameter.

If the specified dump options include subpools for storage areas to be dumped, up to seven subpools can be dumped. Subpool areas are accumulated and wrapped, so that the eighth subpool area specified replaces the first.

If the dump options specified include ranges of storage areas to be dumped, only the storage areas in the first thirty ranges will be dumped.

The TCB, DCB, ID, and STRHDR options available on SNAP or SNAPX are ignored if they appear in the parameter list. The TCB used is the one for the task that encountered the error. The DCB used is one created by the system, and either SYSABEND, SYSMDUMP, or SYSUDUMP is used as a DDNAME.

,REASON=*code*

Specifies the reason code that the user wishes to pass to subsequent recovery routines.

,RC=0
,RC=4
,RC=16

Specifies the return code the recovery routine sends to recovery processing to indicate what further action is required:

0 Continue with error processing, causes entry into previously specified recovery routine, if any.

4 Retry using the retry address specified.

16 Valid only for an ESTAI/STAI recovery routine. The system should not give control to any further ESTAI/STAI routines, and should abnormally end the task.

,RETADDR=*retry addr*
Specifies the address of the retry routine to which control is to be given.

,REMREC=YES
,REMREC=NO
In an ESTAE environment, specifies that the ESTAE entry for the currently running ESTAE routine be removed (REMREC=YES) or not removed (REMREC=NO). This parameter may be specified only when RC=4 is specified, indicating a retry request.

The entry is removed before control returns to the retry point. If REMREC=YES is not coded on any SETRP invocation before the system receives control, the effect is that of specifying REMREC=NO. The REMREC parameter may be used to remove a recovery routine that has been established with a token, although the token cannot be specified when you code the SETRP macro.

,RETREGS=NO
,RETREGS=YES
,RETREGS=YES,RUB=*reg info addr*
,RETREGS=64
Specifies the contents of the registers to be restored on entry to the retry routine. RETREGS=NO indicates that you do not want the system to restore any register contents from the SDWA.

If you specify RETREGS=YES, in a recovery routine defined through the ESTAE or ESTAEX macro, the ESTAI parameter on the ATTACH or ATTACHX macro, or an associated recovery routine (ARR), the system does the following:
- Initializes GPRs 0-15 from the SDWASRSV field of the SDWA
- Initializes ARs 0-15 from the SDWAARSV field of the SDWA.

Specifying RETREGS=64 is the same as specifying RETREGS=YES, except the registers for retry are the 64-bit general purpose registers in field SDWAG64.

RUB (register update block) specifies the address of an area that contains register update information for the GPRs. The data you specify in this area will be moved into the SDWASRSV field of the SDWA and will be loaded into the GPRs on entry to the retry routine. You cannot use the RUB to specify data to be moved into the SDWAARSV field for loading the ARs. The maximum length of the RUB is 66 bytes. You must acquire storage for and initialize this area as follows:
- The first two bytes represent the registers to be updated, register 0 corresponding to bit 0, register 1 corresponding to bit 1, and so on. The user indicates which of the registers are to be stored in the SDWA by setting the corresponding bits in these two bytes.
- The remaining 64 bytes contain the update information for the registers, in the order 0-15. If all 16 registers are being updated, this field consists of 64 bytes. If only one register is being updated, this field consists of only 4 bytes for that one register.

For example, if only registers 4, 6, and 9 are being updated:
- Bits 4, 6, and 9 of the first two bytes are set.

- The remaining field consists of 12 bytes for registers 4, 6, and 9; the first 4 bytes are for register 4, followed by 4 bytes for register 6, and 4 final bytes for register 9.

,FRESDWA=NO
,FRESDWA=YES
Specifies that the entire SDWA be freed (YES) or not be freed (NO) prior to entry into the retry routine.

,COMPCOD=*comp code*
,COMPCOD=(*comp code***,USER)**
,COMPCOD=(*comp code***,SYSTEM)**
Specifies the user or system completion code that the user wishes to pass to subsequent recovery routines.

,RECPARM=*record list addr*
Specifies the address of a user-supplied record parameter list used to update the SDWA with recording information. The parameter list consists of three 8-byte fields:

- The first field contains the load module name.
- The second field contains the CSECT name (assembly module name).
- The third field contains the recovery routine name (assembly module name). If the recovery routine label is not the same as the assembly module name, the label can be used.

The three fields are left-justified, and padded with blanks.

ABEND Codes

None.

Return and Reason Codes

None.

Example 1

Request to continue terminating, suppress dumping, restore register 14 from the save area, and pass control to the location it contains, contain the SDWA in the location addressed by register 3, and change the completion code to 10.

```
SETRP   RC=0,DUMP=NO,REGS=(14),WKAREA=(3),            X
        COMPCOD=(X'00A',USER)
```

Example 2

Retry using address X, take a dump before retry, use the contents of SDWASRSV to initialize the registers, free the SDWA before control is passed to the retry address, and restore registers 14-12.

```
SETRP   RC=4,RETREGS=YES,DUMP=YES,FRESDWA=YES,        X
        REGS=(14,12),RETADDR=X
```

SNAP and SNAPX — Dump Virtual Storage and Continue

Description

You can use the SNAP macro to obtain a dump of some or all of the storage assigned to the current job step. You can also dump some or all of the control program fields. The SNAP macro causes the specified storage to be displayed in the addressing mode of the caller.

Descriptions of the SNAP and SNAPX macros in this book are:

- The standard form of the SNAP macro, which includes general information about the SNAP and SNAPX macros, with some specific information about SNAP. The topic also describes the syntax of the SNAP macro and explains the SNAP macro parameters.
- The standard form of the SNAPX macro, which presents specific information about the SNAPX macro. The topic describes the syntax of the SNAPX macro and explains the parameters that are valid only on the SNAPX macro.
- The list form of the SNAP and SNAPX macros.
- The execute form of the SNAP and SNAPX macros.

There are three ways to obtain a dump:

1. Spool the dump by specifying SYSOUT=x on the DD statement. The dump is printed without a separate job but is deferred until after the job ends.
2. Select a tape or direct access device. This method requires a separate job step to print the dump. This method might be used if the dump is to be printed more than once.
3. Select a printer on the DD statement. This method is almost never used because the printer cannot be used by anyone else for the duration of the job step.

Both NUC and ALLVNUC are valid. Only ALLVNUC gives you the whole virtual nucleus. For more information about the SNAP macro, see *z/OS MVS Programming: Assembler Services Guide*.

Note

The SNAP and SNAPX macros have the same environment specifications, register information, programming requirements, restrictions and limitations, performance implications, and return codes described below. However, **IBM recommends** that programs in access register (AR) address space control (ASC) mode use SNAPX. All parameters on SNAP are valid on SNAPX.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key
Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 24- or 31-bit

© Copyright IBM Corp. 1988, 2002

ASC mode:	Primary or AR **Note:** If your program is in AR mode and you issue SNAP rather than SNAPX following SYSSTATE ASCENV=AR, the system substitutes the SNAPX macro and issues a message telling you that it made the substitution.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held, and no enabled, unlocked task (EUT) FRRs established
Control parameters:	Must be in the primary address space

Input Register Information

Before issuing the SNAP(X) 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
2-14	Unchanged
15	Return code

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

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

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

Programming Requirements

Before you issue the SNAP macro, you must open the DCB that you designate on the DCB parameter, and ensure that the DCB is not closed until the SNAP macro returns control. To open the DCB, issue the DCB macro with the following parameters, and issue an OPEN macro for the data set (the DCB and OPEN macros are described in *MVS/DFP Macro Instructions for Data Sets*):

```
DSORG=PS,RECFM=VBA,MACRF=(W),BLKSIZE=nnn,LRECL=xxx,
and DDNAME=any name but SYSABEND, SYSMDUMP or SYSUDUMP
```

If a standard dump of 120 characters per line is requested, BLKSIZE must be either 882 or 1632, and LRECL must be 125. A high-density dump printed on a 3800 Printing Subsystem has 204 characters per line. To obtain a high-density dump, you must code CHARS=DUMP on the DD statement describing the dump data set. The BLKSIZE= must be either 1470 or 2724, and the LRECL= must be 209. You can also code CHARS=DUMP on the DD statement describing a dump data set that will not be printed immediately. If you specify CHARS=DUMP and the output device is not a 3800, print lines are truncated and print data is lost. If you open a SNAP data set in a problem program that will be processed by the system loader, your problem program must close the data set.

The DCB and TCB must reside in 24-bit addressable storage. All other parameters can reside above 16 megabytes if the issuer is executing in 31-bit addressing mode.

If the program is in AR mode, issue SNAPX rather than SNAP; issue the SYSSTATE ASCENV=AR macro before SNAPX. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

None.

Performance Implications

None.

Syntax

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

---

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

DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,TCB=*tcb addr*	*tcb addr*: A-type address, or register (2) - (12).
,ID=*id nmbr*	*id nmbr*: Symbol, decimal digit, or register (2) - (12). **Value range**: 0-255

,SDATA=ALL
,SDATA=(*sys data code*)
	sys data code: Any combination of the following, separated by commas. If you specify only one code, you do not need the parentheses.
	`NUC CB ERR`
	`SQA Q IO`
	`LSQA TRT ALLVNUC`
	`PCDATA`
	`SWA DM SUM`

,PDATA=ALL ,PDATA=(*prob data code*)	*prob data code*: Any combination of the following, separated by commas. If you specify only one code, you do not need the parentheses.

	PSW REGS SA or SAH JPA or LPA or ALLPA SPLS SUBTASKS
,STORAGE=(*strt addr,end addr*) ,LIST=*list addr*	*strt addr*: A-type address, or register (2) - (12). *end addr*: A-type address, or register (2) - (12). *list addr*: A-type address, or register (2) - (12). **Note**: One or more pairs of addresses may be specified, separated by commas. For example: STORAGE=(*strt addr,end addr, strt addr,end addr*)
,STRHDR=*(hdr addr)* ,STRHDR=*hdr list addr*	*hdr addr*: A-type address, or register (2) - (12). **Note**: *hdr addr* is one or more addresses separated by commas. If you specify only one header address as an A-type address, you do not need the parentheses. If you specify one or more registers, then you must code double parentheses (one set enclosing each register and one set enclosing the list of registers). If STRHDR=(*hdr addr*) is specified, then STORAGE must also be specified. *hdr list addr*: A-type address, or register (2) - (12). **Note**: If STRHDR=*hdr list addr* is specified, then LIST must also be specified.
,SUBPLST=*sbp list addr*	*sbp list addr*: A-type address, or register (2) - (12).

Parameters

The parameters are explained as follows:

DCB=*dcb addr*

Specifies the address of a previously opened data control block for the data set that is to contain the dump.

Notes:

1. DCB must reside in 24-bit addressable storage.
2. The DCB parameter **is not** required when you issue the list form of SNAP or SNAPX to format a parameter list for the DUMPOPT/DUMPOPX parameter of the ABEND, CALLRTM, or SETRP macros. If the parameter list you specify on DUMPOPT/DUMPOPX contains a DCB value, the system overrides it. The DCB parameter **is** required when you issue the list form of SNAP or SNAPX to format a parameter list for an execute form of SNAP or SNAPX if the execute form does not specify the DCB parameter. That is, if you specify both a list and execute form of SNAP or SNAPX, you must specify DCB on one or the other.

,TCB=*tcb addr*

Specifies the address of a fullword on a fullword boundary containing the address of the task control block for a task of the current job step. If omitted, or if the fullword contains 0, the dump is for the active task. If a register is designated, the register can contain 0 to indicate the active task, or can contain the address of a TCB.

Note: TCB must reside in 24-bit addressable storage.

,ID=*id nmbr*
Specifies the number that is to be printed in the identification heading with the dump. If the number specified is not in the acceptable value range, it will not be printed properly in the heading.

,SDATA=ALL
,SDATA=(*sys data code***)**
Specifies the system control program information to be dumped:

ALL All of the SDATA options except ALLVNUC (The read-only portion of the nucleus is not included in the dump unless ALLVNUC is also specified as an option.)

NUC The PSA, SQA, LSQA, and the read/write portion of the nucleus (if the entire nucleus is required, specify the ALLVNUC option.)

Note: The CVT will be included if this option is specified.

SQA The system queue area (subpools 226, 239, and 245).

LSQA The local system queue area and subpools 229, 230, and 249.

Note: Subpools 229, 230, and 249 will be dumped only for the current task.

SWA The scheduler work area related to the task (subpools 236 and 237).

CB The control blocks for the task.

Q The global resource serialization control blocks for the task.

TRT The GTF trace and system trace data. If system tracing is active and the requestor is authorized, all system trace entries for all address spaces are included in the dump. Unauthorized requestors obtain those system trace entries, after the job-start time stamp in the ASCB, for their current address space. If GTF tracing is active, only the GTF trace entries for the current address space are included in the dump.

DM Data management control blocks for the task.

ERR Recovery/termination control blocks for the task. These control blocks summarize information that describes abnormal terminations of the task.

IO Input/Output supervisor control blocks for the task.

ALLVNUC
The entire virtual nucleus, the PSA, LSQA, and SQA. (The NUC option will not dump the read-only section of the nucleus.) If the SNAP parameter list is used for a SYSMDUMP, the ALLVNUC option is converted to ALLNUC on the SVC dump parameter list.

Note: The CVT is included if this option is specified.

PCDATA
Program call information for the task.

The SUM option is valid for an abending task or on a list form of the SNAP macro pointed to by the DUMPOPT keyword of the ABEND or SETRP macro. The option SUM causes the dump to contain a summary dump. If SUM is the only option requested, the dump contains a dump header, control blocks, and the other areas listed below. The header information, which is provided for all ABEND dumps, consists of the following information:

- The dump title

- The ABEND code and program status word (PSW) at the time of the error
- If the PSW contains the address of an active load module:
  - The name and PSW address of the load module in error
  - The offset, into the load module, at which the error occurred

The following control blocks and areas are also included in the dump:

- The control blocks dumped for the CB option
- The error control blocks (RTM2WAs and SCBs)
- The save areas
- The registers at the time of the error, except for register 1
- The contents of the load module (if the PSW contains the address of an active load module)
- The module pointed to by the last PRB (if it can be found)
- 1K of storage before and after the addresses pointed to by the PSW and the registers at the time of the error

  Note: This storage will only be dumped if the caller is authorized to obtain it. The storage is printed by ascending storage addresses with duplicate addresses removed.
- System trace entries after the job-start time stamp in the ASCB for the current address space

  Note: The GTF trace records are not included.

If other options are specified with SUM, the summary dump is dispersed throughout the dump.

,PDATA=ALL
,PDATA=(*prob data code***)**

Specifies the problem program information to be dumped:

ALL All of the following fields.

PSW Program status word when the SNAP or ABEND macro was issued.

REGS Contents of the floating-point registers and general-purpose registers when the SNAP or ABEND macro was issued. Also, contents of the vector registers, vector status register, and the vector mask register when the SNAP or ABEND macro was issued for any task that uses the Vector Facility.

SA Save area linkage information, program call linkage information, and a back trace through save areas.

SAH Save area linkage information and program call linkage information.

JPA Contents of job pack area.

LPA Contents of active link pack area for the requested task.

ALLPA
Contents of job pack area and active link pack area for the requested task.

SPLS Virtual storage subpools 0-127, 131-132, 252.

SUBTASKS
The designated task and the program data information for all of its subtasks.

,STORAGE=(*strt addr,end addr***)**
,LIST=*list addr*

Specifies one or more pairs of starting and ending addresses or a list of starting and ending addresses of areas to be dumped. Each starting address is rounded down to a fullword boundary; each ending address is rounded up to a fullword boundary. The area is then dumped in fullword increments. Callers executing in either 24-bit or 31-bit addressing mode must set the high-order bit of the fullword containing the last address in this list to 1. Callers executing in 31-bit addressing mode must ensure that this bit is cleared in all other addresses in the list because SNAP processing truncates the list at the first address that contains a 1 in the high order bit.

,STRHDR=(*hdr addr***)**
,STRHDR=*hdr list addr*

Specifies one or more header addresses or the address of a list of header addresses. Each header address must be the address of a one byte header length field, which is followed by the text of the header. The header has a maximum length of 100 characters.

If the STORAGE parameter was specified, the STRHDR (storage header) value must be one or more header addresses. The number of pairs of starting and ending addresses specified for STORAGE must be the same as the number of header addresses specified for STRHDR. If a header is not desired for a storage area, a comma must be used to indicate its absence.

If the LIST parameter was specified, the STRHDR value must be the address of a list of header addresses. The list of addresses must begin on a fullword boundary, and the high order bit of the fullword containing the last address of the list must be set to 1. The number of pairs of starting and ending addresses supplied with the LIST parameter must be the same as the number of addresses in the list supplied with STRHDR. If a header is not desired for a storage area, the STRHDR list must contain a zero address to indicate its absence.

,SUBPLST=*sbp list addr*

Specifies the address of a list of subpool numbers to be dumped. Each entry in the list must be a two-byte entry and must specify a valid subpool number. The first halfword of the list must contain the number of subpools in the list and must be on a fullword boundary. If you specify an invalid subpool number or a subpool number for which you do not have authorization, the number is skipped and you receive a comment in the dump output indicating the error. If a subpool contains 4k blocks of data that are mapped from a linear data set, the dump includes only the blocks that have changed since the last DIV SAVE function was invoked.

Note: A maximum of seven subpool numbers is permitted on the list form of the SNAP macro pointed to by the DUMPOPT keyword of ABEND or SETRP.

Return and Reason Codes

Control is returned to the instruction following the SNAP macro. When control is returned, register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion.

Hexadecimal Code	Meaning
04	Data control block was not open, or an invalid page exception occurred during the validity check of the DCB parameters.
08	Task control block address was not valid, an invalid page reference occurred during the validity check of the TCB address, a subtask is a job step task, sufficient storage was not available, or the READ for JFCB or JFCBE failed. In all cases, the dump is canceled. (Message IEA997I is issued when the READ for JFCB or JFCBE fails.) **Or**, the ALET for SNAP parameter list or the ALETs for areas pointed to by the parameter list are not valid.
0C	Data control block type (DSORG, RECFM, MACRF, BLKSIZE, or LRECL) was incorrect, or the DCB's BLKSIZE and/or LRECL were not compatible with the dump format options specified on the dump-related DD statement.

Example 1

Dump the storage ranges pointed to by register 9, and dump all PDATA and SDATA options.

```
SNAP   DCB=(8),TCB=(5),PDATA=ALL,SDATA=ALL,LIST=(9)
```

Example 2

Dump the storage ranges pointed to by register 9, and dump only the trace table and enqueue control blocks.

```
SNAP   DCB=(8),TCB=(5),ID=4,LIST=(9),SDATA=(TRT,Q)
```

Example 3

Dump storage area 1000-2000 with no header, and dump storage area 3000-4000 with a header of 'USER LABEL ONE'. The comma specified in the value for STRHDR indicates that no header is wanted for storage area 1000-2000.

```
       SNAP   DCB=(8),STORAGE=(1000,2000,3000,4000),   X
              STRHDR=(,L1)
       .
       .
       .
L1     DC     AL1(L'HDR1)
HDR1   DC     C'USER LABEL ONE'
```

Example 4

Dump storage area 1000-1999 with a header of 'LABEL ONE' and dump storage area 3000-3999 with a header of 'LABEL TWO'.

```
       SNAP   DCB=(8),LIST=X,STRHDR=L1
       .
       .
       .
X      DC  A(1000)          Start address
       DC  A(1999)          End address
       DC  A(3000)          Start address
       DC  X'80'            End of list indicator
       DC  AL3(3999)        End address
L1     DC  A(HDR1)          Address of length label for
                            header one
       DC  X'80'            End of list
       DC  AL3(HDR2)        Address of length label for
```

```
                                 header two
HDR1   DC  AL1(L'HDR1A)     Length of header one
HDR1A  DC  C'LABEL ONE'     Header one
HDR2   DC  AL1(L'HDR2A)     Length of header two
HDR2A  DC  C'LABEL TWO'     Header two
```

Example 5

Dump subpool 0, 1, and 2 storage related to the current TCB.

```
SNAP DCB=XYZ,TCB=0,SUBPLST=SUBADDR
           .
           .
           .
SUBADDR DS OF          Fullword boundary
        DC X'0003'     Number of entries in the list
        DC X'0000'     Subpool 0
        DC X'0001'     Subpool 1
        DC X'0002'     Subpool 2
```

SNAPX — Dump Virtual Storage and Continue

The SNAPX macro performs the same function as SNAP: it enables you to obtain a dump of some or all of the storage assigned to the current job step. SNAPX is intended for use by programs running in access register (AR) mode. Programs running in primary mode can also use SNAPX.

Note

The SNAPX macro has the same environment specifications, register information, programming requirements, restrictions and limitations, performance implications and return codes as the SNAP macro. However, **IBM recommends** that programs in AR ASC mode use SNAPX. All parameters on SNAP are valid on SNAPX.

Syntax

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

---

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

DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,TCB=*tcb addr*	*tcb addr*: A-type address, or register (2) - (12).
,ID=*id nmbr*	*id nmbr*: Symbol, decimal digit, or register (2) - (12). **Value range**: 0-255

,SDATA=ALL

,SDATA=(*sys data code*)

sys data code: Any combination of the following, separated by commas. If you specify only one code, you do not need the parentheses.

```
NUC       CB        ERR
SQA       Q         IO
LSQA      TRT       ALLVNUC
PCDATA
SWA       DM        SUM
```

,PDATA=ALL

,PDATA=(*prob data code*)

prob data code: Any combination of the following, separated by commas. If you specify only one code, you do not need the parentheses.

```
PSW
REGS
SA or SAH
JPA or LPA or ALLPA
SPLS
SUBTASKS
```

,STORAGE=(*strt addr,end addr*)

strt addr: A-type address, or register (2) - (12).
end addr: A-type address, or register (2) - (12).

,LIST=*list addr*

list addr: A-type address, or register (2) - (12).

Note: One or more pairs of addresses may be specified, separated by commas. For example: STORAGE=(*strt addr,end addr,strt addr,end addr*)

,STRHDR=*(hdr addr)*

hdr addr: A-type address, or register (2) - (12).

,STRHDR=*hdr list addr*

Note: *hdr addr* is one or more addresses separated by commas. If you specify only one header address as an A-type address, you do not need the parentheses. If you specify one or more registers, then you must code double parentheses (one set enclosing each register and one set enclosing the list of registers). If you specify STRHDR=(*hdr addr*), you must also specify STORAGE.

hdr list addr: A-type address, or register (2) - (12).

Note: If you specify STRHDR=*hdr list addr*, you must also specify LIST.

,SUBPLST=*sbp list addr*

sbp list addr: A-type address, or register (2) - (12).

,DSPSTOR=*list addr*

list addr: A-type address or reg (2) - (12).

Parameters

Parameters for the SNAPX macro are the same as those for the SNAP macro, except for the DSPSTOR parameter, which is valid only on SNAPX. SDATA=SUM has a different function for callers in AR mode. These two parameters are described as follows:

,SDATA=SUM

The SUM option is valid for an abending task or on a list form of the SNAPX macro pointed to by the DUMPOPX parameter of the ABEND or SETRP macro. For the contents of the summary dump, see the description of the SDATA parameter in the SNAP macro.

,DSPSTOR=*list addr*

Specifies the address of a list of data space storage areas to be dumped. Use this parameter to dump data that is in a data space.

Each entry in the parameter list you create describes an area to be dumped; the entry must contain a start address, end address, and STOKEN. The list must begin on a fullword boundary, and the high order bit of the fullword containing the last end address in the list must be set to 1. The system dumps storage from any data space to which the caller has authority; it does not dump storage to which the caller does not have authority.

You can specify the DSPSTOR parameter for SNAPX parameter lists that are identified by the DUMPOPX parameter on the ABEND or SETRP macro.

SNAP and SNAPX—List Form

Use the list form of the SNAP or SNAPX macro to construct a control program parameter list. You can specify any number of storage addresses using the STORAGE parameter. Therefore, the number of starting and ending address pairs in the list form of SNAP or SNAPX must be equal to the maximum number of addresses specified in any execute form of the macro, or a DS instruction must immediately follow the list form to allow for the maximum number of addresses.

Syntax

The list form of the SNAP or SNAPX macro is written as follows:

---

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

DCB=*dcb addr*	*dcb addr*: A-type address. **Note:** The DCB parameter **is not** required in all cases. See the parameter description for details.
,ID=*id nmbr*	*id nmbr*: Symbol or decimal digit. **Value range**: 0-255
,SDATA=ALL ,SDATA=(*sys data code*)	*sys data code*: Any combination of the following, separated by commas. If you specify only one code, you do not need parentheses. NUC CB ERR SQA Q IO LSQA TRT ALLVNUC PCDATA SWA DM SUM
,PDATA=ALL ,PDATA=(*prob data code*)	*prob data code*: Any combination of the following, separated by commas. If you specify only one code, you do not need parentheses.

	PSW REGS SA or SAH JPA or LPA or ALLPA SPLS SUBTASKS
,STORAGE=(*strt addr,end addr*) ,LIST=*list addr*	*strt addr*: A-type address. *end addr*: A-type address. *list addr*: A-type address. **Note**: One or more pairs of addresses may be specified, separated by commas. For example: STORAGE=(*strt addr,end addr,strt addr,end addr*)
,STRHDR=(*hdr addr*) ,STRHDR=*hdr list addr*	*hdr addr*: A-type address. **Note**: hdr addr is one or more addresses separated by commas. If you specify only one header address, you do not need the parentheses. If STRHDR=(*hdr addr*) is specified, then STORAGE must also be specified. *hdr list addr*: A-type address. **Note**: If STRHDR=*hdr list addr* is specified, then LIST must also be specified.
,SUBPLST=*sbp list addr*	*sbp list addr*: A-type address.
,DSPSTOR=*list addr*	*list addr*: A-type address or register (2) - (12).

,MF=L

Parameters

The parameters are explained under the standard form of the SNAP and SNAPX macros, with the following exception:

,MF=L
Specifies the list form of the SNAP or SNAPX macro.

SNAP and SNAPX—Execute Form

A remote control-program parameter list is referred to and can be modified by the execute form of the SNAP or SNAPX macro.

If you code only the DCB, ID, MF, or TCB parameters in the execute form of the macro, the bit settings in the parameter list corresponding to the SDATA, PDATA, LIST, and STORAGE parameters are not changed. However, if you code the SDATA, PDATA, or LIST parameters, the bit settings for the coded parameter from the previous request are reset to zero, and only the areas requested in the current macro are dumped.

Syntax

The execute form of the SNAP or SNAPX macro is written as follows:

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

ƀ — One or more blanks must precede SNAP.

SNAP
SNAPX

ƀ — One or more blanks must follow SNAP.

DCB=*dcb addr* — *dcb addr*: RX-type address, or register (2) - (12).
Note: The DCB parameter **is not** required in all cases. See the parameter description for details.

,TCB=*tcb addr*
,TCB=‘S’ — *tcb addr*: RX-type address, or register (2) - (12).

,ID=*id nmbr* — *id nmbr*: Symbol, decimal digit or register (2) - (12).
Value range: 0-255

,SDATA=ALL
,SDATA=(*sys data code*) — *sys data code*: Any combination of the following, separated by commas. If you specify only one code, you do not need parentheses.

```
NUC      CB      ERR
SQA      Q       IO
LSQA     TRT     ALLVNUC
PCDATA
SWA      DM      SUM
```

,PDATA=ALL
,PDATA=(*prob data code*) — *prob data code*: Any combination of the following, separated by commas. If you specify only one code, you do not need parentheses.

```
PSW
REGS
SA or SAH
JPA or LPA or ALLPA
SPLS
SUBTASKS
```

,STORAGE=(*strt addr,end addr*) — *strt addr*: RX-type address, or register (2) - (12).
end addr: RX-type address, or register (2) - (12).
,LIST=*list addr* — *list addr*: RX-type address, or register (2) - (12).
Note: One or more pairs of addresses may be specified, separated by commas. For example:
STORAGE=(*strt addr,end addr,strt addr,end addr*)

,STRHDR=(*hdr addr*)
,STRHDR=*hdr list addr* — *hdr addr*: RX-type address, or register (2) - (12).
Note: *hdr addr* is one or more addresses separated by commas. If you specify only one header address as an RX-type address, you do not need the parentheses. If you specify one or more registers, then you must code double parentheses (one set enclosing each register and one set enclosing the list of registers). If STRHDR=*(hdr addr)* is specified, then STORAGE must also be specified.

	hdr list addr: RX-type address, or register (2) - (12). **Note:** If STRHDR=*hdr list addr* is specified, then LIST must also be specified.
,SUBPLST=*sbp list addr*	*sbp list addr*: RX-type address, or register (2) - (12).
,DSPSTOR=*list addr*	*list addr*: A-type address or register (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 the SNAP and SNAPX macros, with the following exceptions:

,TCB=‘S’
Specifies the task control block of the active task.

Note: TCB=‘S’ causes a dump of the active task if this is the first use of the list form of the SNAP or SNAPX macro or if the TCB specified on a previous execute form of the SNAP or SNAPX macro was the current TCB or TCB=‘S’.

,MF=(E,*ctrl addr*)
specifies the execute form of the SNAP or SNAPX macro using a remote control program parameter list.

SPIE — Specify Program Interruption Exit

Description

Note: IBM recommends that you use the ESPIE macro rather than SPIE. Callers in 31-bit addressing mode must use the ESPIE macro, which performs the same function as the SPIE macro for callers in both 24-bit and 31-bit addressing mode.

The SPIE macro specifies the address of an interruption exit routine and the program interruption types that are to cause the exit routine to get control.

Note: In MVS/370 the SPIE environment existed for the life of the task. In later versions of MVS, the SPIE environment is deleted when the request block that created it is deleted. That is, when a program running under a later version of MVS completes, any SPIE environments created by the program are deleted. This might create an incompatibility with MVS/SP Version 1 for programs that depend on the SPIE environment remaining in effect for the life of the task rather than the request block.

Each succeeding SPIE macro completely overrides any previous SPIE macro specifications for the task. The specified exit routine is given control in the key of the TCB when one of the specified program interruptions occurs in any problem program of the task. When a SPIE macro is issued from a SPIE exit routine, the program interruption element (PIE) is reset (zeroed). Thus, a SPIE exit routine should save any required PIE data before issuing a SPIE. If a caller issues an ESPIE macro from within a SPIE exit routine, it has no effect on the contents of the PIE. However, if an ESPIE macro deletes the last SPIE/ESPIE environment, the PIE is freed and the SPIE exit cannot retry.

If the current SPIE environment is cancelled during SPIE exit routine processing, the control program will not return to the interrupted program when the SPIE program terminates. Therefore, if the SPIE exit routine wishes to retry within the interrupted program, a SPIE cancel should not be issued within the SPIE exit routine.

The SPIE macro can be issued by any problem program being executed in the performance of the task. The control program automatically deletes the SPIE exit routine when the request block (RB) that issued the SPIE macro terminates.

A PICA (program interruption control area) is created as part of the expansion of SPIE. The PICA contains the exit routine's address and a code indicating the interruption types specified in SPIE.

For more information on the SPIE macro, see the section on program interruption services in *z/OS MVS Programming: Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization: To issue SPIE without encountering an abnormal end, callers must be in problem state, with a PSW key value that is equal to the TCB assigned key.

Dispatchable unit mode: Task

© Copyright IBM Corp. 1988, 2002

Cross memory mode:	PASN=HASN=SASN
AMODE:	24-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 must include the following mapping macros:

- IHAPIE
- IHAPICA

Restrictions

None.

Input Register Information

Before issuing the SPIE 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	If a SPIE environment is already active when you issue the SPIE macro, the SPIE service routine returns the address of the previous PICA in register 1. You can use this PICA to restore the previously active SPIE environment. However, if an ESPIE environment is active when you issue the SPIE macro, the SPIE service returns the address, in register 1, of a PICA in which the first word contains binary zeros. You cannot modify the contents of this PICA, and it contains no useful information except to restore the previous SPIE or ESPIE environment. If no previous SPIE/ESPIE environment is active, the service routine returns a zero in register 1.
2-13	Unchanged.
14-15	Used as work registers 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 SPIE macro is written as follows:

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

exit addr	*exit addr*: A-type address, or register (2) - (12).
,(interrupts)	*interrupts*: Decimal numbers 1-15 expressed as: single values: (2,3,4,7,8,9,10) ranges of values: ((2,4),(7,10)) combinations: (2,3,4,(7,10))

Parameters

The parameters are explained as follows:

exit addr
Specifies the address of the exit routine to be given control when a specific program interruption occurs. The exit routine receives control in 24-bit addressing mode.

,(*interrupts***)**
Indicates the type of interruption for which the exit routine is to be given control. The interruption types are as follows:

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

Notes:

1. If an exit address is zero or no parameters are specified, the current SPIE and any previously active ESPIE environments are cancelled.
2. If a program interruption type is maskable, the corresponding program mask bit in the PSW (program status word) is set to 1 when specified and to 0 when not specified. Interruption types that are not maskable and not specified above are handled by the system, which forces an abend with the program check as the completion code. If an ESTAE-type recovery routine is also active, the SDWA indicates a system-forced abnormal termination. The registers at the time of the error are those of the system.
3. 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.

ABEND Codes

The SPIE macro might return abend codes X'10E', X'30E', or X'46D'. See *z/OS MVS System Codes* for explanations and programmer responses.

Return and Reason Codes

None.

Example

Give control to an exit routine for interruption 1, 5, 7, 8, 9, and 10. DOITSPIE is the address of the SPIE exit routine.

```
SPIE DOITSPIE,(1,5,7(8,10))
```

SPIE—List Form

Use the list form of the SPIE macro to construct a control program parameter list in the form of a program interruption control area.

Syntax

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

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

exit addr	*exit addr*: A-type address.

,(interrupts)	*interrupts*: Decimal numbers 1-15 expressed as: . single values: (2,3,4,7,8,9,10) ranges of values: ((2,4),(7,10)) combinations: (2,3,4,(7,10))
,MF=L	

Parameters

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

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

SPIE—Execute Form

A remote control program parameter list is used in, and can be modified by, the execute form of the SPIE macro. The PICA (program interruptions control area) can be generated by the list form of SPIE, or you can use the address of the PICA returned in register 1 following a previous SPIE macro. If this macro is being issued to reestablish a previous SPIE environment, code only the MF parameter.

Syntax

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

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

exit addr	*exit addr*: RX-type address, or register (2) - (12).
,(interrupts)	*interrupts*: Decimal numbers 1-15, expressed as single values: (2,3,4,7,8,9,10) ranges of values: ((2,4),(7,10)) combinations: (2,3,4,(7,10))
,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 SPIE macro, with the following exception:

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

Note: If SPIE is coded with a 0 as the control address, the SPIE environment is canceled.

SPLEVEL — Set Macro Level

Description

Use the SPLEVEL macro to ensure that the assembler generates the correct level for a particular macro that your program issues. You might need to control the level of a macro expansion if you assemble your program on one version and release of MVS, then run the program on a different version and release of MVS, and one of the following is true:

- Your program issues MVS macros that are **downward incompatible** to MVS/System Product Version 1.
- Your program issues installation- or vendor-written macros that are incompatible between versions and releases.

See “Compatibility of MVS Macros” on page 1 for additional information about the downward incompatible MVS macros. Authorized callers of SPLEVEL should consult “Selecting the Macro Level” in the following for the lists of downward incompatible MVS macros that are authorized:

- *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 installation- or vendor-written macros, see the installation or vendor documentation to determine if incompatibilities between versions and releases exist.

You can use SPLEVEL in two ways:

- Within your program, issue SPLEVEL with the SET=*n* parameter prior to issuing another macro to set the desired level for that macro. SPLEVEL SET=*n* sets a global symbol (&SYSSPLV) to the value *n*. Certain macros (including all the downward incompatible macros) check this global symbol during assembly to determine which expansion of the macro to generate. Once you set the macro level, all macros in your program that check the &SYSSPLV global symbol expand at that level until you change the level to some other value.

  See Table 3 on page 17 for the list of macros that check the SPLEVEL global symbol. Authorized callers of SPLEVEL should consult the Macro Summary in the chapter entitled “Using the Macros” in the following publications for the lists of authorized macros that check the SPLEVEL global symbol:

  – *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*

  See *High Level Assembler Language Reference* for information about global set symbols.
- Within a macro you are writing, issue SPLEVEL with the TEST parameter to ensure that the macro level is set:
  1. Define the &SYSSPLV global symbol within your macro.
  2. Issue SPLEVEL TEST, which checks to see if the caller set the macro level.

© Copyright IBM Corp. 1988, 2002

3. Define different logical paths within your macro to correspond to the macro level that is in effect.

Existing programs that were assembled using Version 2, Version 3, Version 4, and Version 5 macros will run properly on OS/390 and z/OS. z/OS, OS/390 and version 5 macros will run properly on OS/390 systems without your issuing the SPLEVEL macro.

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 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

None.

Restrictions

None.

Input Register Information

Before issuing the SPLEVEL 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) and access registers (ARs) are all unchanged.

Performance Implications

None.

Syntax

The SPLEVEL macro is written as follows:

---

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

ƀ — One or more blanks must precede SPLEVEL.

SPLEVEL

ƀ — One or more blanks must follow SPLEVEL.

TEST	
SET=*n*	*n*: 2, 3, 4, 5 or 6
SET	**Default**: SET=6

Parameters

The parameters are explained as follows:

TEST
TEST checks the &SYSSPLV global variable, and does the following:

- Sets &SYSSPLV to the default value if &SYSSPLV **does not** contain a value indicating that you did not issue SPLEVEL SET during this assembly, .
- Leaves the value of &SYSSPLV unchanged, if &SYSSPLV **does** contain a value indicating that you issued SPLEVEL SET during this assembly.

SET=*n*
SET
Specifies the macro level by setting the global symbol &SYSSPLV.

- SET=*n* places a value in &SYSSPLV equal to *n*, where *n* must be 2, 3, 4, 5 or 6.
- SET without *n*, results in the assembler using the default value, 6.

If you then issue a macro that checks the &SYSSPLV global symbol, the assembler generates one of the following macro expansions:

- The MVS/SP Version 1 Release 3 macro expansion if *n*=1
- The MVS/SP Version 2 macro expansion if *n*=2
- The MVS/SP Version 3 macro expansion if *n*=3
- The MVS/ESA SP Version 4 macro expansion if *n*=4
- The OS/390 Release 1 and MVS/ESA SP Version 5 macro expansion if *n*=5
- The OS/390 Release 2 macro expansion if *n*=6

ABEND Codes

None.

Return and Reason Codes

None.

Example 1

Select the version 1 expansion of a specific downward incompatible macro.

```
SPLEVEL SET=1
```

Example 2

Use SPLEVEL TEST within your own macro to ensure the &SYSSPLV global symbol is set.

```
.
.
.
```

```
        GBLC     &SYSSPLV                 Define global symbol
        SPLEVEL  TEST                     If global symbol has no value,
                                          set to the default.
        AIF      ('&SYSSPLV' EQ '1').V1   Use code for V1
.V5     ANOP     This logical path contains instructions appropriate
                 for a V2, V3, V4, or V5 expansion.
        .
        .
        .
        AGO      .COMMON
.V1     ANOP     This logical path contains instructions appropriate
                 for a V1 expansion.
        .
        .
        .
.COMMON ANOP
```

STAE — Specify Task Abnormal Exit

Note: IBM recommends that you use the ESTAEX macro or ESTAE macro rather than STAE.

Description

The STAE macro enables the user to intercept a scheduled ABEND and to have control returned to him at a specified exit routine address. The STAE macro operates in both problem program and supervisor modes.

Note: The STAE macro is not supported for users executing in 31-bit addressing mode. Such users will be abended.

Syntax

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

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

0 *exit addr*	*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=QUIESCE ,PURGE=HALT ,PURGE=NONE	**Default**: PURGE=QUIESCE
,ASYNCH=NO ,ASYNCH=YES	**Default**: ASYNCH=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.

© Copyright IBM Corp. 1988, 2002

Parameters

The parameters are explained as follows:

0
exit addr
Specifies the address of a STAE exit routine to be entered if the task issuing this macro terminates abnormally. If 0 is specified, the most recent STAE request is canceled.

,CT
,OV
Specifies the creation of a new STAE exit (CT) or indicates that the parameters passed in this STAE macro are to overlay the data contained in the previous STAE exit (OV).

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

,XCTL=NO
,XCTL=YES
Specifies that the STAE macro will be canceled (NO) or will not be canceled (YES) if an XCTL macro is issued by this program.

,PURGE=QUIESCE
,PURGE=HALT
,PURGE=NONE
Specifies that all outstanding requests for I/O operations are not saved when the STAE exit is taken (HALT), that I/O processing is allowed to continue normally when the STAE exit is taken (NONE), or that all outstanding requests for I/O operations are saved when the STAE exit is taken (QUIESCE). For QUIESCE, at the end of the STAE exit routine, the user can code a retry routine to handle the outstanding I/O requests.

Note: If any IBM-supplied access method, except EXCP, is being used, the PURGE=NONE option is recommended. If you use PURGE=NONE, all control blocks affected by input/output processing can continue to change during STAE exit routine processing.

If PURGE=NONE is specified and the ABEND was originally scheduled because of an error in input/output processing, an ABEND recursion develops when an input/output interruption occurs, even if the exit routine is in progress. Thus, it appears that the exit routine failed when, in reality, input/output processing caused the failure.

ISAM Notes: If ISAM is being used and PURGE=HALT is specified or PURGE=QUIESCE is specified but I/O is not restored:

- Only the input/output event on which the purge is done is posted. Subsequent event control blocks (ECBs) are not posted.
- The ISAM check routine treats purged I/O as normal I/O.
- Part of the data set may be destroyed if the data set is being updated or added to when the failure occurred.

,ASYNCH=NO
,ASYNCH=YES
Specifies that asynchronous exit processing is allowed (YES) or is not allowed (NO) while the STAE exit is executing.

ASYNCH=YES must be coded if:

- The STAE exit routine requests any supervisor services that require asynchronous interruptions to complete their normal processing.
- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE=NONE is specified and the CHECK macro is issued in the STAE exit routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

Note: If ASYNCH=YES is specified and the ABEND was originally scheduled because of an error in asynchronous exit handling, an ABEND recursion develops when an asynchronous interruption occurs. Thus, it appears that the exit routine failed when, in reality, asynchronous exit handling caused the failure.

,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. Control returns to the instruction following the STAE macro; register 15 contains one of the following return codes:

Hexadecimal Code	Meaning
00	Successful completion of STAE request.
04	STAE was unable to obtain storage for STAE request.
08	Attempt was made to cancel or overlay a nonexistent STAE request.
0C	Exit routine or parameter list address was invalid, or STAI request was missing a TCB address.
10	Attempt was made to cancel or overlay a STAE request of another user, or an unexpected error was encountered while processing this request.

Example

Request an overlay of the existing STAE recovery exit with the following options: new exit address is ADDR, parameter list is at PLIST, halt I/O, do not take asynchronous exits, transfer ownership to the new request block resulting from any XCTL macros.

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

STAE—List Form

The list form of the STAE macro is used to construct a remote control program parameter list.

Syntax

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

---

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

STAE

b	One or more blanks must follow STAE.

---

exit addr	*exit addr*: A-type address.
,PARAM=*list addr*	*list addr*: A-type address.
,PURGE=QUIESCE ,PURGE=HALT ,PURGE=NONE	**Default**: PURGE=QUIESCE
,ASYNCH=NO ,ASYNCH=YES	**Default**: ASYNCH=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.

,MF=L

Parameters

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

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

STAE—Execute Form

A remote control program parameter list is used in, and can be modified by, the execute form of the STAE macro. The control program parameter list can be generated by the list form of the STAE macro. If you want to dynamically change the contents of the remote STAE parameter list, you can do so by coding a new exit address and/or a new parameter list address. If exit address or PARM= is coded, only the associated field in the remote STAE parameter list is changed. The other field remains as it was before the current STAE request was made.

Syntax

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

---

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

STAE

b	One or more blanks must follow STAE.

exit addr	*exit addr*: RX-type address, or register (2) - (12).
0	
,CT	
,OV	
,PARAM=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,XCTL=NO	
,XCTL=YES	
,PURGE=QUIESCE	
,PURGE=HALT	
,PURGE=NONE	
,ASYNCH=NO	
,ASYNCH=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,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 STAE macro, with the following exception:

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

Example

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

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

STATUS — Start and Stop a Subtask

Description

Use the STATUS macro to change the dispatchability status of one or all of a program's subtasks. For example, the STATUS macro can be used to restart subtasks that were stopped when an attention exit routine was entered.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	No requirements.

Programming Requirements

None.

Restrictions

The caller cannot have an EUT FRR established.

Input Register Information

Before issuing the STATUS 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.

© Copyright IBM Corp. 1988, 2002

Performance Implications

Using STATUS will degrade performance of the calling program's address space while STATUS runs.

Syntax

The STATUS macro is written as follows:

---

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

START STOP	
,TCB=*tcb addr*	*tcb addr*: RX-type address or address in register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

START
STOP

Specifies that the task identified on the TCB parameter is to be stopped (STOP) or started (START). If you omit the TCB parameter, all subtasks of the originating task are stopped or started.

Note: This parameter does not ensure that the subtask is stopped when control is returned to the issuer. A subtask can have a "stop deferred" condition that would cause that particular subtask to remain dispatchable until stops are no longer deferred. In a multiprogramming environment, it would be possible to have a task issue the STATUS macro with the STOP parameter and resume processing while the subtask (for which the STOP was issued) is redispatched to another processor.

,TCB=*tcb addr*

Specifies the address of a fullword on a fullword boundary containing the address of the task control block that is to have its START/STOP count adjusted. (If a register is specified, however, the address is of the TCB itself.) If this parameter is not coded, the count is adjusted in the task control blocks for all the subtasks of the originating task.

Note: TCB must reside in 24-bit addressable storage.

,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.

The RELATED parameter is available on macros that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE) and on macros that relate to previous occurrences of the same macros (for example, CHAP and ESTAE).

The RELATED parameter may be used, for example, as follows:

```
STAT1   STATUS    STOP,TCB=YOURTCB,RELATED=(STAT2,
                  'STOP A SUBTASK')
  .
  .
  .
STAT2   STATUS    START,TCB=YOURTCB,RELATED=(STAT1,
                  'START A SUBTASK')
```

Note: Each of these macros will fit on one line when coded, so there is no need for a continuation indicator.

Return Codes

Return codes from execution of STATUS are as follows:

Table 29. Return Codes for the STATUS Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Processing completed successfully. **Action**: No action necessary.
04	**Meaning**: Program error. START/STOP request failed. The task you specified is not a subtask of the calling program’s task. **Action**: Ensure that you specify a task on the TCB parameter that is a subtask of the calling program.

Example 1

Stop all subtasks.

```
STATUS   STOP
```

Example 2

Create a subtask. Stop the subtask, then restart it.

```
         PRINT NOGEN
STATUS   CSECT
STATUS   AMODE 31
STATUS   RMODE ANY
***********************************************************************
*  The following code performs the following functions:               *
*      1.  Creates a subtask by issuing the ATTACH macro.             *
*      2.  Stops the subtask by issuing the STATUS macro with the     *
*          STOP parameter.                                            *
*      3.  Starts the stopped subtask by issuing the STATUS macro     *
*          with the START parameter.                                  *
*                                                                     *
***********************************************************************
         SPACE 3
```

```
*************************************
* Entry linkage                     *
*************************************
         SPACE 3
         STM   R14,R12,12(R13)
         BALR  R12,0
         USING BEGN,R12
BEGN     DS    0H
         ST    R13,SAVE+4
         LA    R15,SAVE
         ST    R15,8(0,R13)
         LR    R13,R15
         EJECT
*********************************************************************
* Attach a subtask and request that it be notified by an ECB when    *
* the subtask completes.                                             *
*                                                                    *
*********************************************************************
         SPACE 3
ATTCH1   ATTACH EP=SUBTASK,ECB=AMYECB
         SPACE 3
         ST     R1,TCBADDR         SAVE SUBTASK TCB ADDRESS
         EJECT
*****************************************************************
*  Stop the subtask by issuing STATUS STOP, then restart it by   *
*  issuing STATUS START.                                         *
*                                                                *
*****************************************************************
         SPACE 3
         STATUS STOP,TCB=TCBADDR
         SPACE 3
         .
****************************************************
* Processing of other subtasks continues.         *
****************************************************
         .
         STATUS START,TCB=TCBADDR
         SPACE 3
         EJECT
*****************************************************
* Wait until subtask completes, then detach it.     *
*****************************************************
         SPACE 3
         WAIT  1,ECB=AMYECB        WAIT ON E-O-T ECB
         SPACE 3
         DETACH TCBADDR           DETACH SUBTASK
         SPACE 3
         EJECT
*************************************
* End of job                        *
*************************************
         SPACE 3
FINI     DS    0H
         L     R13,SAVE+4
         DROP  R12
         LM    R14,R12,12(R13)
         XR    R15,R15
         BR    R14
         EJECT
***********************************
*   Define constants              *
***********************************
SAVE     DC    18F'0'
*
TCBADDR  DC    F'0'        ADDRESS OF SUBTASK TCB
AMYECB   DC    F'0'        END-OF-SUBTASK ECB
         EJECT
```

```
***********************************************
*   Register equates                          *
***********************************************
         SPACE 3
R1       EQU   1
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         LTORG
         END
```

STCKCONV — Store Clock Conversion Routine

Description

The STCKCONV macro converts an input time-of-day (TOD) clock value to time of day and date, and returns the converted values to the caller in the format requested. The input clock value can be either the basic time-of-day (TOD) format or the extended time-of-day (ETOD) format.

- TOD — Unsigned 64-bit binary number
- ETOD — Unsigned 128-bit binary number

See *z/OS MVS Programming: Assembler Services Guide* and *z/Architecture Principles of Operation* for information comparing the formats of the TOD and ETOD.

The STCKCONV time of day and date formats are compatible with the formats returned by the TIME macro, which returns a time of day and date value or the contents of the TOD clock. The STCKCONV time of day and date formats are also compatible with the input formats accepted by the CONVTOD macro, which converts a time of day and date value to TOD clock format.

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 or PASN¬=HASN¬=SASN
AMODE:	24-bit or 31-bit addressing mode
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
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 the program is in AR mode, issue the SYSSTATE ASCENV=AR macro before STCKCONV. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

None.

Input Register Information

Primary-mode callers must make sure that access register 1 is zero before issuing the execute form of the STCKCONV macro. For other registers, 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:

© 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 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 STCKCONV macro is written as follows:

---

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

STCKVAL=*TOD clock addr*	*TOD clock addr*: RX-type address or register (2) - (12).
STCKEVAL=*ETOD clock addr*	*ETOD clock addr*: RX-type address or register (2) - (12).
,CONVVAL=*conv addr*	*conv addr*: RX-type address or register (2) - (12).
,TIMETYPE=DEC ,TIMETYPE=BIN ,TIMETYPE=MIC	**Default**: TIMETYPE=DEC
,DATETYPE=YYYYDDD ,DATETYPE=DDMMYYYY ,DATETYPE=MMDDYYYY ,DATETYPE=YYYYMMDD	**Default**: DATETYPE=YYYYDDD

Parameters

The parameters are explained as follows:

STCKVAL=*TOD clock addr*
Specifies the address of an 8-byte storage area containing the 64-bit TOD clock value to be converted.

STCKEVAL=*ETOD clock addr*
Specifies the address of a 16-byte storage area containing the 128-bit ETOD clock value to be converted.

Only one of STCLVAL or STCKEVAL can be specified.

,CONVVAL=*conv addr*
Specifies the address of a 16-byte storage area where the system returns the converted value in the requested format. The first two words contain the time of day and the third word contains the date. Do not use the contents of the fourth word.

,TIMETYPE=DEC
,TIMETYPE=BIN
,TIMETYPE=MIC
Specifies the format in which the converted time of day is returned, as follows:

DEC Returns the converted time of day as packed decimal digits (without a sign) of the form HHMMSSthmiju0000, where
- **HH** is hours, based on a 24-hour clock
- **MM** is minutes
- **SS** is seconds
- **t** is tenths of a second
- **h** is hundredths of a second
- **m** is milliseconds
- **i** is ten-thousandths of a second
- **j** is hundred-thousandths of a second
- **u** is microseconds

BIN Returns the converted time of day as an unsigned 32-bit binary number with the low-order bit equivalent to 0.01 second. The second word of the converted time value is zero.

MIC Returns the converted time of day in microseconds as 8 bytes of information, where bit 51 is equivalent to one microsecond.

,DATETYPE=YYYYDDD
,DATETYPE=DDMMYYYY
,DATETYPE=MMDDYYYY
,DATETYPE=YYYYMMDD
Specifies the format in which the converted date is returned, as follows:

Parameter	Form of returned date
YYYYDDD	0YYYYDDD
DDMMYYYY	DDMMYYYY
MMDDYYYY	MMDDYYYY
YYYYMMDD	YYYYMMDD

The date is returned as 4 bytes of packed decimal digits (without a sign), where:
YYYY is the year

DDD	is the day of the year
DD	is the day of the month
MM	is the month of the year

ABEND Codes

None.

Return Codes

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

Table 30. Return Codes for the STCKCONV Macro

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: Successful completion. **Action**: None.
C	**Meaning**: System error. **Action**: Retry the request.
10	**Meaning**: Program error. The user's parameter list is not in addressable storage. **Action**: Ensure that the parameter list address is valid and the storage is addressable.

Example 1

Convert a TOD clock value to time of day in decimal digits, and date in month-day-year format.

```
         STCKCONV STCKVAL=TODSTAMP,CONVVAL=OUTAREA,TIMETYPE=DEC,        X
               DATETYPE=MMDDYYYY
TODSTAMP DC X'A0569832F1241000'     TOD CLOCK VALUE
OUTAREA  DS CL16                    CONVERTED VALUE
```

Example 2

Convert a TOD clock value to time of day in hundredths of seconds, and date in year-month-day format.

```
         STCK TODCLOCK
         STCKCONV STCKVAL=TODCLOCK,CONVVAL=OUTVAL,TIMETYPE=BIN,         X
               DATETYPE=YYYYMMDD
TODCLOCK DS XL8       TOD CLOCK VALUE
OUTVAL   DS CL16      CONVERTED VALUE
```

STCKCONV—List Form

Use the list form of the STCKCONV 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.

Syntax

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

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

MF=L

Parameter

The parameter is explained as follows:

MF=L
Specifies the list form of the STCKCONV macro. Do not specify any other keywords with MF=L. Precede the STCKCONV list form macro invocation with a name starting in column 1 to label the generated parameter list so you can refer to it.

Example

Establish the correct amount of storage for the STCKCONV parameter list.

```
LIST1    STCKCONV MF=L
```

STCKCONV—Execute Form

Use the execute form of the STCKCONV 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 STCKCONV macro is written as follows:

---

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

STCKVAL=*TOD clock addr*	*TOD clock addr*: RX-type address or register (2) - (12).
STCKEVAL=*ETOD clock addr*	*ETOD clock addr* RX-type address or register (2) - (12).

,CONVVAL=*conv addr*	*conv addr*: RX-type address or register (2) - (12).
,TIMETYPE=DEC ,TIMETYPE=BIN ,TIMETYPE=MIC	**Default**: TIMETYPE=DEC
,DATETYPE=YYYYDDD ,DATETYPE=DDMMYYYY ,DATETYPE=MMDDYYYY ,DATETYPE=YYYYMMDD	**Default**: DATETYPE=YYYYDDD
,MF=(E,*list addr*)	*list addr*: RX-type address or register (1) - (12).

Parameters

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

,MF=(E,*list addr*)
Specifies the execute form of the STCKCONV macro. *list addr* specifies the address of the parameter list created by the list form of the macro.

Example

Convert a TOD clock value to time of day in microseconds and date in year-day of the year format. Specify the address of the appropriate parameter list in LIST1.

```
          STCKCONV STCKVAL=TODCLOCK,CONVVAL=OUTVAL,TIMETYPE=MIC,        X
                DATETYPE=YYYYDDD,MF=(E,LIST1)
TODCLOCK  DC X'9FE4781301ABE000'     TOD CLOCK VALUE
OUTVAL    DS CL16                    CONVERTED VALUE
```

STCKSYNC — Store Clock Synchronous Service

Description

The STCKSYNC macro obtains the time-of-day (TOD) clock contents and indicates whether the TOD clock is synchronized with an external time reference (ETR[2]). It is for use by programs that are dependent upon synchronized TOD clocks in a multisystem environment. STCKSYNC also provides an optional parameter, ETRID, that returns the network ID of the ETR source with which the TOD clock is synchronized.

The time-of-day clock specified can be either the basic time-of-day clock format (TOD) or the extended time-of-day clock format (ETOD).

- TOD — Unsigned 64-bit binary number
- ETOD — Unsigned 128=bit binary number

See *z/OS MVS Programming: Assembler Services Guide* or *z/Architecture Principles of Operation* for information comparing the formats of the TOD and ETOD.

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 or PASN¬=HASN¬=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	Any locks may be held, no locks required
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 the program is in AR mode, issue the SYSSTATE ASCENV=AR macro before STCKSYNC. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

None.

Input Register Information

For primary ASC mode callers, GPR 13 must contain the address of a 72-byte save area. For AR mode callers, AR/GPR 13 must contain the address of a 72-byte save area.

Output Register Information

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

2. External time reference (ETR) is the MVS generic name for the IBM Sysplex Timer.

© 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 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 STCKSYNC macro is written as follows:

---

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

TOD =*TOD clock addr*	*TOD clock addr*: RX-type address
ETOD=*ETOD clock addr*	*ETOD clock addr*: RX-type address
,ETRID=*id addr*	*id addr*: RX-type address

Parameters

The parameters are explained as follows:

TOD=*TOD clock addr*
Specifies the address of a doubleword that receives the TOD clock value.

ETOD=*ETOD clock addr*
Specifies the address of a 16-byte area, aligned on a double-word boundary, that receives the extended TOD clock value (ETOD).

Only one of either TOD or ETOD can be specified.

,ETRID=*id addr*
Specifies the address of a byte that receives the ETR network ID of the ETR with which the TOD clock is synchronized. No ETRID value is returned if the TOD clock is not synchronized with an ETR.

ABEND Codes

None.

Return Codes

Return codes from the STCKSYNC macro are returned as hexadecimal values in register 15, as follows:

Table 31. Return Codes for the STCKSYNC Macro

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: The TOD clock is synchronized with an ETR or a simulated ETR was requested (through SYS1.PARMLIB member CLOCKxx). If ETRID was specified, the ID of the ETR is returned at *id addr*. **Action**: None.
4	**Meaning**: The TOD clock is not synchronized with an ETR. **Action**: None required. However, you might take some action based upon your application.
8	**Meaning**: System error. The TOD clock is unusable. **Action**: Reissue the request until it succeeds.

Example 1

Obtain the TOD clock contents and an indication of whether the TOD clock is synchronized with an ETR.

```
         STCKSYNC TOD=TODAREA
TODAREA  DS XL8                    TOD CLOCK CONTENTS
```

Example 2

For a caller in AR mode, obtain the TOD clock contents, an indication of whether the TOD clock is synchronized with an ETR, and the network ID of the ETR source with which the TOD clock is synchronized.

```
         SYSSTATE ASCENV=AR
         .
         .
         .
         STCKSYNC TOD=TODAREA,ETRID=IDAREA
TODAREA  DS XL8            TOD CLOCK CONTENTS
IDAREA   DS XL1            ETR NET ID
```

STIMER — Set Interval Timer

Description

The STIMER macro sets a timer to a specified time interval or to an interval that will expire at a specified time of day. An optional asynchronous timer completion exit is given control when the time interval expires; if no asynchronous timer completion routine is specified, no indication that the time interval has expired is provided. A second STIMER macro issued before the first time interval expires overrides the first interval and exit routine.

The time interval may be a ‘real-time interval’ (measured continuously in real time by the clock comparator), or a ‘task-time interval’ (measured, only while the task is in execution, by the CPU timer). See *Principles of Operation* for information on the clock comparator and CPU timer. If a real-time interval is specified, the task may elect to either continue (REAL) or suspend (WAIT) execution during the interval. If the task elects to continue execution, it may optionally specify an exit routine to be given control on completion of the time interval. If the task elects to suspend execution, it is restarted at the next sequential instruction, sometime after completion of the time interval. If a task-time interval is specified, the task must continue. It may optionally specify an exit routine to be given control on completion of the interval.

STIMER allows you to set one time interval for one task; STIMERM allows you to set 16 separate time intervals for a task. Using the two macros together allows you to set 17 separate intervals for a task.

For information on how to select an MVS/SP version other than the current version, see “Compatibility of MVS Macros” on page 1. If your program is to execute in 31-bit addressing mode, you must use the SP Version 2 expansion of this macro or a later version.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key.
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

The timer completion exit routine must be in virtual storage when it is required.

Restrictions

The following restrictions apply to the STIMER macro:

- Only one STIMER invocation can be active at a time. Ensure that any processing your program performs after issuing the STIMER macro does not also invoke the STIMER macro. For concurrent requests, use the STIMERM macro.

© Copyright IBM Corp. 1988, 2002

- Do not issue the STIMER macro while a BTAM OPEN or LINE OPEN operation is in progress. Use STIMERM instead.
- Do not issue the STIMER macro before invoking dynamic allocation. Use STIMERM instead.
- For REAL or WAIT requests:
  - If you specify a time of day at which the interval will expire (GMT (Greenwich Mean Time), LT (local time), or TOD (Time of Day) parameters), the time of day you specify must not exceed 24:00:00:00; otherwise, your program receives a X'12F' abend.
  - If you specify a time interval on the MICVL parameter, the interval you specify, when added to the current TOD clock contents, must not exceed the maximum value for the clock comparator (X'FFFFFFFFFFFFFFFF'); otherwise, your program receives a X'12F' abend.
- For TASK requests, the time interval you specify on MICVL must not exceed the maximum positive value for the CPU timer (X'7FFFFFFFFFFFFFFF'); otherwise, your program receives a X'12F' abend.
- You can issue STIMER REAL with a timer completion exit routine, and within that routine, you can issue STIMER REAL and specify the same timer completion exit routine. Under these circumstances, IBM recommends that you specify a time interval rather than a time of day on the STIMER you issue within the timer completion exit routine. If you specify a time of day, it is possible for the timer completion exit routine to receive control later than the time of day you specified, resulting in an infinite loop.
- The caller can have no enabled, unlocked task (EUT) FRRs established.
- The time interval you specify on the BINTVL parameter must not exceed X'7FFFFFFF'. If the time interval exceeds X'7FFFFFFF', your program receives a X'12F' abend.
- If you make use of JES2 main task exit routines or have vendor code that could run under the JES2 main task, then this code *cannot* use the STIMER macro. Such use would usurp the timer JES2 sets with its use of the STIMER macro. The exit or vendor code would destroy JES2 processing and lead to unpredictable errors. STIMERM is the macro this code must use instead.

Input Register Information

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

Output Register Information

When control returns to the caller, the registers 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	0 (zero)

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 STIMER macro is written as follows:

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

REAL REAL,*exit rtn addr* TASK TASK,*exit rtn addr* WAIT	*exit rtn addr*: RX-type address, or register (0) or (2) - (12).
,BINTVL=*stor addr* ,DINTVL=*stor addr* ,MICVL=*stor addr* ,GMT=*stor addr* ,TUINTVL=*stor addr* ,TOD=*stor addr* ,LT=*stor addr*	*stor addr*: RX-type address, or register (1) or (2) - (12). **Note**: The GMT, TOD, and LT parameters must not be specified with TASK above.

Note: The ERRET parameter is obsolete and is ignored by the system. Therefore, the syntax and parameter descriptions for STIMER no longer contain ERRET. However, the system still accepts ERRET, and it is not necessary to delete it from existing code.

Parameters

The parameters are explained as follows:

REAL
REAL,*exit rtn addr*
TASK
TASK,*exit rtn addr*
WAIT

Specifies whether the timer interval is a real-time interval (REAL or WAIT) or a task-time interval (TASK). You must specify one of these parameters.

For REAL, the interval is decreased continuously. If the TOD, GMT, or LT parameter is coded, the interval expires at the indicated time of day.

For TASK, the interval is decreased only when the associated task is running.

For WAIT, the interval is decreased continuously. The task is to be placed in the wait condition until the interval expires.

The *exit rtn addr* is the address of the timer completion exit routine to be given control after the specified time interval expires. The routine does not get control immediately when the interval completes, but at some time after the interval completes, depending on the system's work load and the relative dispatching priority of the associated task. The routine must be in virtual storage when it is required. The exit routine receives control in the same environment that the caller had when the caller issued the STIMER macro. The contents of the registers when the exit routine is given control are as follows:

Register	Contents
0-12	Do not contain any information for use by the routine.
13	Address of a system-provided, 72-byte save area.
14	Return address (to the system).
15	Address of the exit routine.

The exit routine is responsible for saving and restoring registers. The exit routine runs as a subroutine, and must return control to the address identified in register 14. Although timing services allows only one active time interval for a task, it does not serialize the use of an asynchronous timer completion exit routine.

,BINTVL=*stor addr*
,DINTVL=*stor addr*
,GMT=*stor addr*
,MICVL=*stor addr*
,TOD=*stor addr*
,TUINTVL=*stor addr*
,LT=*stor addr*

Specifies the storage address and format for the time of day, or time interval, to be set. You must specify one of these parameters.

For BINTVL, the address is a 4-byte area containing the time interval. The time interval is represented as an unsigned 32-bit binary number; however, the high-order bit of the time interval must not be set. Therefore, the time interval specified cannot exceed X'7FFFFFFF'. The low-order bit of the time interval has a value of 0.01 second.

For DINTVL, the address is a doubleword in virtual storage containing the time interval. The time interval 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

For GMT, the address is an 8-byte area containing the Greenwich mean time at which the interval is to be completed. The time is presented as zoned decimal digits of the form HHMMSSth, as described above under DINTVL.

For MICVL, the address is a doubleword containing the time interval. The time interval is represented as an unsigned 64-bit binary number; bit 51 is the low-order bit of the interval value and equivalent to 1 microsecond.

For TUINTVL, the address is a fullword containing the time interval. The time interval is presented as an unsigned 32-bit binary number; the low-order bit has a value of one timer unit (approximately 26.04166 microseconds).

For TOD and LT, the address is a doubleword containing the local time of day at which the interval is to be completed. The time is presented as zoned decimal digits of the form HHMMSSth, as described under DINTVL.

The LT and TOD parameters perform identical functions. However, the name for the LT parameter (LT, or local time) describes the function more accurately than does the name for the TOD parameter (TOD, or time-of-day). Therefore, for clarity purposes, IBM recommends the use of the LT parameter instead of TOD.

Note: For the DINTVL, GMT, TOD, and LT parameters, the zoned decimal digits are not checked for validity. Thus, the specification of incorrect digits can result in an X'0C7' abend, or a time interval different from that desired.

Notes:

1. The time interval specified by an STIMER macro has no relation to the time interval specified in an EXEC statement.
2. If no exit routine address is specified, there is no indication of completion except when WAIT is specified.
3. The TTIMER and CPUTIMER macros provide a facility for determining the remaining time interval associated with STIMER.

The priorities of other tasks in the system can also affect the accuracy of the time interval measurement. If you code REAL or WAIT, the interval is decreased continuously and can expire when the task is not active. After the time interval expires, assuming the task is not in the wait condition for any other reasons, the task is placed in the ready condition and competes for control with the other ready tasks in the system. The additional time required before the task becomes active depends on the relative dispatching priority of the task.

ABEND Codes

STIMER might abnormally terminate with one the following abend codes: X'12F' (with reason code X'0', X'4', X'C', X'10', X'14', X'28'), or X'AC7' (with reason code X'2'). See *z/OS MVS System Codes* for an explanation and response for these codes.

Return and Reason Codes

STIMER returns a return code of 0 in register 15.

Examples

Example 1: Request the installation's asynchronous exit routine, located at location EXIT, to receive control after fourteen hundredths of a second (specified by INTVLONG) have elapsed in real time.

```
         STIMER   REAL,EXIT,BINTVL=INTVLONG
⋮
INTVLONG DC X'0000000E'     TIME INTERVAL
```

Example 2: Request that this task's exit routine, located at location EXIT, receive control when the local time of day specified at location LOCAL occurs.

```
         STIMER   REAL,EXIT,LT=LOCAL
⋮
LOCAL    DS       2F
```

Example 3: Request that this task be put into a wait state until 60 seconds have passed.

```
         STIMER   WAIT,DINTVL=INTV2
⋮
         DS       2F
INTV2    DC X'F0F0F0F0F6F0F0C0'
```

Example 4: Request that this task's exit routine, located at location EXIT, receive control when the task has executed 60 seconds.

```
         STIMER   TASK,EXIT,BINTVL=INTV1
⋮
INTV1    DS       F
         DC X'00001770'
```

STIMERM — Set, Test, Cancel Multiple Interval Timer

Description

The STIMERM macro:
- Sets a timer to a specified time interval (SET parameter)
- Tests the remaining time interval for a timer request (TEST parameter)
- Cancels a specific timer request (CANCEL parameter)

The SET request sets a timer to a specified time interval or to an interval that will expire at a specified time of day. Up to sixteen STIMERM requests per task may be in effect at a time.

The time interval is a real-time interval, measured continuously. The task can continue (WAIT=NO) or suspend execution (WAIT=YES). If the task continues execution, it can pass control to an exit routine (EXIT parameter) when the time interval is complete. If you specify an exit routine, the task can optionally pass a parameter to the exit routine (PARM parameter). The task grants control to the optional asynchronous timer completion exit when the time interval expires. If the task did not specify either an asynchronous timer completion routine or WAIT=YES, the task receives no indication that the time interval has expired.

The TEST request tests the remaining time interval for a timer request established through the SET parameter. The ID parameter identifies the particular timer request to be tested and must be established by the current task.

The CANCEL request cancels a specific timer request or all of the current task's timer requests that were established through the SET parameter. The ID parameter identifies the timer request or requests to be cancelled. If the macro cancels a specific timer request, it may return the remaining time interval for that request to a storage area designated by the TU (Timer Units) or MIC (Microseconds) parameters.

On the TEST and CANCEL requests, the TU and MIC parameters specify the location where the system returns the remaining time:
- If you specify TU, the STIMERM macro returns the amount of time remaining to the designated 4-byte storage area as an unsigned 32-bit binary number containing the number of timer units (approximately 26.04166 microseconds per unit) remaining in the interval.
- If you specify MIC, the STIMERM macro returns the remaining time to the designated 8-byte storage area. Bit 51 of the area is the low-order bit of the interval value and is equivalent to approximately one microsecond.

If the specified timer request does not exist for the current task, or if the timer request exists but has expired, the system sets to zero the storage area designated by TU or MIC.

When you cancel a timer request that specified a timer exit, specify TU or MIC to determine whether the cancel operation was successful:
- If STIMERM returned a value of zero to the storage area designated by TU or MIC, then any associated timer exit has run or will run because its interval expired before the cancel operation completed.

© Copyright IBM Corp. 1988, 2002

- If STIMERM returned a non-zero value to the storage area designated by TU or MIC, then the timer interval was cancelled and any associated timer exit will not run.

It is your responsibility to set up your program to determine whether the timer exit has run. For information about interval timing, see *z/OS MVS Programming: Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key.
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 or external interrupts
Locks:	No locks held.
Control parameters:	Must be in the primary address space.

Programming Requirements

- All input and output addresses are treated as full 31-bit addresses.
- The parameter lists may be above or below 16 megabytes.
- There is no interaction between the TTIMER macro support and the STIMERM macro support or between the STIMER macro support and the STIMERM macro support.
- If the STIMERM macro service cannot access the macro parameter list or any in-storage parameters, the system abnormally ends the calling program whether or not it specified an ERRET routine.

Restrictions

No enabled, unlocked task (EUT) FRRs may be established.

For SET requests:

- If you specify a time of day at which the interval will expire (GMT, LT, or TOD parameters), the time of day you specify must not exceed 24:00:00.00; otherwise, you receive a X'32E' abend unless you specify ERRET.
- If you specify a time interval on the MICVL parameter, the interval you specify, when added to the current TOD clock contents, must not exceed the maximum value for the clock comparator (X'FFFFFFFFFFFFFFFF'); otherwise, you receive a X'32E' abend unless you specify ERRET.
- The time interval specified by a STIMERM macro has no relation to the time interval specified in an EXEC statement.
- You can issue STIMERM with a timer completion exit routine and, within that routine, you can issue STIMERM REAL and specify the same timer completion exit routine. Under these circumstances, IBM recommends that you specify a time interval rather than a time of day on the STIMERM you issue within the timer completion exit routine. If you specify a time of day, it is possible for the timer completion exit routine to receive control later than the time of day you specified, resulting in a infinite loop.

- The time interval you specify on the BINTVL parameter must not exceed X'7FFFFFFF'. If the time interval exceeds X'7FFFFFFF', your program receives a X'32E' abend unless you use the ERRET parameter to specify a recovery routine.
- No enabled, unlocked task (EUT) FRRs can be established.

TEST and CANCEL requests have no restrictions.

Input Register Information

Before issuing the STIMERM 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 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

Syntax

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

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

	Valid parameters (Required parameters are underlined)
SET	**For SET**: ID, BINTVL or DINTVL or GMT or MICVL or TOD
TEST	or TUINTVL or LT, ERRET, WAIT, EXIT, PARM, RELATED

CANCEL	**For TEST**: ID, TU or MIC, ERRET, RELATED **For CANCEL**: ID, TU or MIC, ERRET, RELATED
,ID=*stor addr* ,ID=ALL	*stor addr*: RX-type address or register (2) - (12). **Note**: ID=ALL is only valid on the CANCEL request.
,TU=*stor addr* ,MIC=*stor addr*	*stor addr*: RX-type address or register (2) - (12).
,BINTVL=*stor addr* ,DINTVL=*stor addr* ,MICVL=*stor addr* ,GMT=*stor addr* ,TUINTVL=*stor addr* ,TOD=*stor addr* ,LT=*stor addr*	*stor addr*: RX-type address or register (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr*: RX-type address or register (2) - (12).
,EXIT=*exit rtn addr*	*exit rtn addr*: RX-type address or register (2) - (12). **Note**: EXIT must not be specified if WAIT=YES is specified.
,PARM=*stor addr*	*stor addr*: RX-type address or register (2) - (12). **Note**: If PARM is specified, EXIT must be specified and WAIT=YES must not be specified.
,WAIT=YES ,WAIT=NO	**Default**: WAIT=NO
,RELATED=*value*	

Parameters

The parameters are explained as follows:

SET
TEST
CANCEL

Request to establish, return, or cancel a real-time interval. You must specify one of these parameters.

SET indicates a request to establish a real-time interval.

TEST indicates a request to return the remaining time for a request made using the SET parameter.

CANCEL indicates a request to cancel and optionally return the remaining time for a timer request.

If the CANCEL parameter specifies (through ID=) a timer request that was established with the WAIT=YES parameter, the task will still remain in the wait condition.

,ID=*stor addr*
,ID=ALL

Specifies the address of a 4-byte area containing the identifier assigned to a

particular timer request by the timer service routine. When you specify STIMER SET, the ID is returned in the 4-byte area. Specify this ID on STIMER TEST or STIMER CANCEL. ID=ALL, valid only on STIMERM CANCEL, cancels all the current task's timer requests as established by STIMERM SET. If you specify ID=ALL, the system does not return a remaining time interval. Do not specify MIC or TU with ID=ALL.

,TU=*stor addr*
,MIC=*stor addr*

Specifies that the remaining time in the interval be returned to the 4-byte or 8-byte area specified in *stor addr*. TU or MIC is required for STIMERM TEST and is optional for STIMERM CANCEL (providing you do not also specify ID=ALL). TU and MIC are mutually exclusive.

For TU, the time is returned to the specified 4-byte area as an unsigned 32-bit binary number. The low-order bit is approximately 26.04166 microseconds (one timer unit). If the time remaining is too great to be expressed in 4 bytes, the remaining time interval is set to the maximum possible value (X'FFFFFFFF') and the return code is set to 4.

For MIC, the time is returned to the specified 8-byte area as microseconds. The 8-byte area stores the remaining interval, which is represented as an unsigned 64-bit binary number; bit 51 is equivalent to one microsecond.

,BINTVL=*stor addr*
,DINTVL=*stor addr*
,GMT=*stor addr*
,MICVL=*stor addr*
,TUINTVL=*stor addr*
,TOD=*stor addr*
,LT=*stor addr*

Specifies the storage address and format of the time of day, or time interval, to be set. You must specify one of these parameters.

For BINTVL, the address is a 4-byte area containing the time interval. The time interval is represented as an unsigned 32-bit binary number; however, the high-order bit of the time interval must not be set. Therefore, the time interval specified cannot exceed X'7FFFFFFF'. The low-order bit of the time interval has a value of 0.01 second.

For DINTVL, the address is an 8-byte area in virtual storage containing the time interval. The time interval is represented as zoned decimal digits of the form:

HHMMSSth, where:

HH	is hours
MM	is minutes
SS	is seconds
t	is tenths of seconds
h	is hundredths of seconds

For GMT, the address is an 8-byte area containing the Greenwich mean time at which the interval will complete. The time is represented as zoned decimal digits of the form HHMMSSth, as described previously under DINTVL.

For MICVL, the address is an 8-byte storage area containing the time interval. The time interval is represented as an unsigned 64-bit binary number; bit 51 is the low-order bit of the interval value and equivalent to one microsecond.

For TUINTVL, the address is a 4-byte area containing the time interval. The time interval is represented as an unsigned 32-bit binary number; the low-order bit has a value of one timer unit (approximately 26.04166 microseconds).

For TOD and LT, the address is an 8-byte storage area containing the local time of day at which the interval is to be completed. The time of day is represented as zoned decimal digits of the form HHMMSSth, as described previously under DINTVL.

The LT and TOD parameters perform identical functions. However, the name for the LT parameter (LT or local time) describes the function more accurately than does the name for the TOD parameter (TOD or time-of-day). Therefore, for clarity purposes, **IBM recommends** the use of the LT parameter instead of TOD.

Notes on setting the time interval: For the DINTVL, GMT, TOD, and LT parameters, the zoned decimal digits are not checked for validity. Thus, specifying invalid digits can cause a X'0C7' abend or an undesired time interval.

,ERRET=*err rtn addr*

Specifies the address of the routine to receive control when the STIMERM function cannot be performed. If you omit this parameter and your program encounters an error, the system abnormally ends your program. The specified error routine will be entered in the addressing mode and environment of the STIMERM invoker.

When the routine receives control, the register contents are:

Register	Contents
0	Address of a 24-byte STIMERM parameter list.
1	Does not contain any information for use by the routine.
2-13	The contents are the same as they were when the caller issued STIMERM.
14	Return address.
15	Return code.

If the macro parameter list or any in-storage parameters are not accessible, the system abnormally ends your program regardless of whether or not you specified ERRET. No error routine will receive control.

,EXIT=*exit rtn addr*

Specifies the address of an exit routine that will gain asynchronous control after the requested timer interval expires. The system's workload and the relative dispatching priority of the associated task determine exactly when, after the interval completes, the exit routine gets control. The specified exit routine will be entered in the addressing mode and environment of the STIMERM invoker. If you specify WAIT=YES, you must *not* specify the EXIT parameter.

Exit Routine Interface

The timer exit routine, established with the EXIT parameter in the STIMERM macro, receives control with the following register values:

R0 -	Does not contain any information for use by the routine
R1 -	Points to an 8-byte fetch-protected storage area below 16 megabytes and in the protect key of the program that issued the STIMERM SET macro

R1 - - - ->	Word 1 TIMER REQUEST ID
	Word 2 USER PARAMETER (specified in the PARM keyword)

R2-R12 - Do not contain any information for use by the routine
R13 - Address of a 72-byte save area provided by the system
R14 - Return address (to the system)
R15 - Address of the exit routine

The exit routine receives control in the addressing mode of the STIMERM issuer. If multiple asynchronous exits are established, the exit routines may not receive control in the same order that the intervals expire.

,PARM=*stor addr*
Specifies the address of a 4-byte parameter that the exit routine receives when the requested timer interval expires. You must *not* specify PARM=*stor addr* if you specified WAIT=YES. If you specify PARM=*stor addr*, you must also specify EXIT=*exit rtn addr*.

An exit routine will be unable to distinguish between the case where PARM= was not specified and the case where the specified PARM value was zero.

,WAIT=YES
,WAIT=NO
Specifies whether the task should be suspended until the requested time interval expires. WAIT=YES specifies that the task should be suspended until the requested time interval expires. If you specify WAIT=NO without specifying EXIT, you will receive no indication when the timer expires. WAIT=NO 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 specified information are at your discretion and may be any valid macro keyword expression.

ABEND Codes

On STIMERM SET requests:

- X'32E'

  Abend code X'32E' might yield the following reason codes:
  - X'10C'
  - X'110'
  - X'11C'
  - X'120'
  - X'128'
- X'AC7'

  Abend code X'AC7' might yield the following reason code:
  - X'2'

On STIMERM TEST requests:

- X'32E'

  Abend code X'32E' might yield the following reason codes:
  - X'210'
  - X'220'
  - X'224'

On STIMERM CANCEL requests:

- X'32E'

  Abend code X'32E' might yield the following reason codes:
  - X'310'
  - X'320'
  - X'324'

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

Return Codes

When control is returned, register 15 contains one of the following hexadecimal return codes. Note that for non-zero return codes, the ERRET routine receives control (if you specified ERRET). If you did not specify ERRET, a non-zero return code causes the STIMERM invoker to end abnormally.

Table 32. Return Codes for the STIMERM Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The STIMERM service has completed successfully. **Action**: None.
04	**Meaning**: For TEST and CANCEL requests, the time remaining is too great to be expressed in 4 bytes. The maximum value (X'FFFFFFFF') is returned. **Action**: None required. However, you might take some action based upon your application.
0C	**Meaning**: Program error. For SET requests, the GMT, LT, or TOD at which the interval is to complete exceeds 24:00:00.00. **Action**: Specify a time of day value that is less than or equal to 2400 hours.
10	**Meaning**: Program error. Parameters passed to STIMERM are not valid. **Action**: Ensure that all input parameters are valid.
1C	**Meaning**: Program error. The request would cause the limit of concurrent STIMERM SET requests for a task to be exceeded. **Action**: Change your application logic so that fewer STIMERM requests are required.
24	**Meaning**: Program error. The specified STIMERM ID number was zero, which is not valid. **Action**: Ensure that the input ID is a valid value.
28	**Meaning**: Program error. For SET requests, either you specified a time interval on the MICVL parameter that, when added to the current TOD clock contents, exceeds the maximum value for the clock comparator (X'FFFFFFFFFFFFFFFF') or you specified a value greater than X'7FFFFFFF' for BINTVL. **Action**: Request a smaller time interval.

Example 1

SET a timer to a specified time interval. Specify:

- The address of a 4-byte area in which the identifier assigned by the timer service to this request will be returned
- That control should be given to an asynchronous timer completion exit named TIME, when the time interval expires
- The address of a 4-byte area (containing the time interval of 32 hundredths of seconds) named INTERVAL. Include an error exit routine named ERROR.

```
         STIMERM SET,ID=ADDRESS,BINTVL=INTERVAL,EXIT=TIME,ERRET=ERROR
ADDRESS  DS F             ID RETURNED
INTERVAL DC X'00000020'   TIME INTERVAL
```

Example 2

SET a timer to a time interval that specifies the address of a 4-byte area in which the identifier assigned by timer service will be returned. Specify the address of an 8-byte area named INTERVAL that contains the Greenwich mean time at which the interval is to be completed (2:06 PM). Specify that the task should be suspended until the requested time interval expires. Include an error exit routine named EXITX.

```
         STIMERM SET,ID=ADDRESS,GMT=INTERVAL,WAIT=YES,ERRET=EXITX
ADDRESS  DS F                     ID RETURNED
INTERVAL DC X'F1F4F0F6F0F0F0F0'   EXPIRATION TIME OF DAY
```

Example 3

SET a timer to a time interval that specifies the address of a 4-byte area in which the identifier assigned by timer service will be returned. Specify the address of an 8-byte area in register 8 that contains the time interval (represented as zoned decimal digits). Specify, in register 10, the address of the exit routine that will gain control asynchronously when the requested time interval expires. Specify the address of a 4-byte parameter to be passed to the exit routine when the requested time interval expires. Include the address of an exit error routine in register 9.

```
         STIMERM SET,ID=(7),DINTVL=(8),PARM=USERDATA,ERRET=(9),EXIT=(10)
USERDATA DC CL4'ABCD'     PARAMETER PASSED TO EXIT ROUTINE
```

Example 4

Test the remaining time interval for a timer request established with the SET parameter, specifying (in register 4) the address of a 4-byte area from which the identifier assigned by the timer service will be obtained. Specify that the time be returned as an unsigned 32-bit binary number in a 4-byte area called INTERVAL. Include the address of an exit error routine called XYZ.

```
         STIMERM TEST,ID=(4),TU=INTERVAL,ERRET=XYZ
INTERVAL DS XL4    REMAINING TIME
```

Example 5

Test the remaining time interval for a timer request established with the SET parameter, specifying the address of a 4-byte area from which the identifier assigned by the timer service will be obtained. Specify that the time be returned in microseconds in an 8-byte area called INTERVAL. Include the address of an exit error routine called ERRORADD.

```
         STIMERM TEST,ID=ADDR,MIC=INTERVAL,ERRET=ERRORADD
ADDR     DS F      ID TO BE TESTED
INTERVAL DS XL8    REMAINING TIME
```

Example 6

Cancel a timer request established with a SET parameter, specifying the address of a 4-byte area named ADDRESS containing the identifier assigned by the timer

service. The time interval remaining should be returned as an unsigned 32-bit binary number in a 4-byte area called INTERVAL. An exit error routine named ERROR is also specified.

```
         STIMERM CANCEL,ID=ADDRESS,TU=INTERVAL,ERRET=ERROR
ADDRESS  DS F      ID TO BE CANCELLED
INTERVAL DS XL4    REMAINING TIME
```

Example 7

Cancel a timer request established with a SET parameter, specifying the address of a 4-byte area named PLACE containing the identifier assigned by the timer service. The time interval remaining should be returned in an 8-byte area called INTERVAL. An exit error routine named EXITA is also specified.

```
         STIMERM CANCEL,ID=PLACE,MIC=INTERVAL,ERRET=EXITA
PLACE    DS F      ID TO BE CANCELLED
INTERVAL DS XL8    REMAINING TIME
```

Example 8

Cancel all the timer requests established with STIMERM SET for the current task.

```
STIMERM CANCEL,ID=ALL
```

STIMERM—List Form

Use the list form of the STIMERM 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 STIMERM macro is written as follows:

---

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

SET
TEST
CANCEL

,MF=L

,RELATED=*value*

Parameters

The parameters are explained as follows:

,MF=L
Specifies the list form of the STIMERM macro. If you do not specify MF=L, the standard form of the macro is expanded. If you do specify MF=L, the only keyword allowed is RELATED.

Example 1

Establish a remote STIMERM SET parameter list.

```
REMOTE   STIMERM SET,MF=L
```

Example 2

Establish a remote STIMERM TEST or CANCEL parameter list.

```
         STIMERM TEST,MF=L
```

Example 3

Establish the appropriate storage for the execute form of the STIMERM CANCEL macro.

```
         STIMERM CANCEL,MF=L
```

STIMERM—Execute Form

Use the execute form of the STIMERM 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 STIMERM macro is written as follows:

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

	Valid parameters (Required parameters are underlined)
SET TEST CANCEL	**For SET**: ID, BINTVL or DINTVL or GMT or MICVL or TOD or TUINTVL or LT, ERRET, WAIT, EXIT, PARM, RELATED **For TEST**: ID, TU or MIC, ERRET, RELATED **For CANCEL**: ID, TU or MIC, ERRET, RELATED
,ID=*stor addr* ,ID=ALL	*stor addr*: A-type address or register (2) - (12). **Note**: ID=ALL is valid only on the CANCEL request.

,TU=*stor addr* ,MIC=*stor addr*	*stor addr*: A-type address or register (2) - (12).
,BINTVL=*stor addr* ,DINTVL=*stor addr* ,GMT=*stor addr* ,MICVL=*stor addr* ,TOD=*stor addr* ,TUINTVL=*stor addr* ,LT=*stor addr*	*stor addr*: A-type address or register (2) - (12).
,ERRET=*err rtn addr*	*err rtn addr*: A-type address or register (2) - (12).
,WAIT=YES ,WAIT=NO	**Default**: WAIT=NO
,EXIT=*exit rtn addr*	*exit rtn addr*: A-type address or register (2) - (12). **Note**: EXIT must not be specified if WAIT=YES is specified.
,PARM=*stor addr*	*stor addr*: A-type address or register (2) - (12). **Note**: If PARM is specified, EXIT must be specified and WAIT=YES must not be specified.
,MF=(E,*ctrl addr*)	*ctrl addr*: A-type address or register (0), (2)-(12) for TEST and CANCEL, register (1)-(12) for SET.
,RELATED=*value*	

Parameters

The parameters are explained in the standard form of the STIMERM macro, with the following exception.

,MF=(E,*ctrl addr*)
Specifies the execute form of the STIMERM macro using a remote problem-program parameter list.

Example 1

Set a timer to a time interval of 15 microseconds, specifying the address of a 4-byte area in which the identifier assigned to this request by timer service will be returned. Specify:

- The address of an 8-byte area in INTERVAL that contains the time interval (represented as an unsigned 64-bit binary number)
- The address of a program to receive asynchronous control after the requested timer interval expires
- The address of a 4-byte parameter to be passed to the exit routine when the requested time interval expires
- The address of the appropriate parameter list in REMOTE

Include the address of an error routine in register 9.

```
         STIMERM SET,ID=(4),MICVL=(INTERVAL),EXIT=ROUTE,PARM=DATA,     X
               MF=(E,REMOTE),ERRET=(9)
DATA     DC CL4'WXYZ'             PARAMETER PASSED TO THE EXIT ROUTINE
INTERVAL DC X'000000000000F000'   TIME INTERVAL
```

Example 2

Test the remaining time interval for a timer request established with the SET parameter, specifying the address of a 4-byte area from which the identifier assigned by timer service will be obtained. Specify that register 3 will point to the appropriate list. Specify that the time be returned in microseconds in an 8-byte area at the address named INTERVAL. Include the address of an exit error routine called ERR.

```
         STIMERM TEST,ID=ADDR,MIC=INTERVAL,MF=(E,(3)),ERRET=ERR
INTERVAL DS XL8     REMAINING TIME
```

Example 3

Cancel the timer request established with a SET parameter. Specify the address of a 4-byte identifier (assigned by timer service) named ADDRESS and that the time interval remaining be returned as an unsigned binary number in a 4-byte area named INTERVAL. Specify that register 0 will point to the appropriate list. Specify an error exit routine named ERROR.

```
         STIMERM CANCEL,ID=ADDRESS,TU=INTERVAL,MF=(E,(0)),ERRET=ERROR
ADDRESS  DS F         ID TO BE CANCELLED
INTERVAL DS XL4       REMAINING TIME
```

STORAGE — Obtain and Release Storage

Description

The STORAGE macro requests that the system obtain or release an area of virtual storage in the primary address space. The two functions of the macro are:

- STORAGE OBTAIN, which obtains virtual storage in an address space
- STORAGE RELEASE, which releases virtual storage in an address space.

If you use STORAGE OBTAIN 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 OS/390 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 OS/390.

Environment

The requirements on the caller are:

Minimum authorization:	For subpools 0-127: problem state and PSW key 8-15. For subpools 131 and 132: a PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be obtained or released.
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-or 64-bit
ASC mode:	Primary or AR
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held.
Control parameters:	No requirement.

Programming Requirements

None.

Restrictions

None.

Register Information

Register usage varies depending on the type of STORAGE request. For specific information, see the descriptions of STORAGE OBTAIN and STORAGE RELEASE.

Performance Implications

None.

OBTAIN Option of STORAGE

The STORAGE macro with the OBTAIN parameter requests that the system allocate an area of virtual storage to the active task. Each virtual storage area begins on a doubleword or page boundary. The amount of storage you request must not exceed the amount available; the amount available depends on how much storage has already been allocated, and on your user region size. Valid subpools

© Copyright IBM Corp. 1988, 2002

available for problem-state callers are 0 - 127, 131, and 132. When a task terminates, the system frees any storage in subpools 0 - 127 that has been allocated to the terminating task. The system does not free storage in subpools 131 and 132 until the job-step task terminates.

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.

Input Register Information

Before issuing the STORAGE macro with the OBTAIN parameter, 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	For a successful request in which maximum and minimum lengths were specified, contains the length of the storage obtained. Otherwise, used as a work register by the system.
1	When STORAGE OBTAIN is successful, 0; otherwise, used as a work register by the system. **Note:** In an AMODE 64 routine, a successful STORAGE OBTAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).
2-13	Unchanged.
14	Used as a work register by the system.
15	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	Used as work registers by the system
1	0 when the STORAGE OBTAIN is successful; otherwise, used as work register 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.

Syntax

The STORAGE macro with the OBTAIN parameter is written as follows:

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

OBTAIN

,LENGTH=*length value*	*length value*: Symbol, decimal number, or register (0), (2) - (12).
,LENGTH=*(max amount,min amount*)	*max amount*: Symbol, decimal number, or register (0), (2) - (12). *min amount*: Symbol, decimal number, or register (1) - (12).
,ADDR=*stor addr*	*stor addr*: RX-type address or register (1) - (12). **Default**: ADDR=(1).
,INADDR=*stor addr*	*stor addr*: RX-type address or register (1)-(12). **Note**: This parameter can only be specified with LOC=EXPLICIT.
,SP=*subpool number*	*subpool number*: Symbol, decimal number 0-127, 131, 132, or register (2) - (12), (15). **Default**: SP=0.
,BNDRY=DBLWD ,BNDRY=PAGE	**Default**: BNDRY=DBLWD
,CONTBDY=*containing_bdy* ,STARTBDY=*starting_bdy*	*containing_bdy*: Decimal number 3-31 or register (2) - (12). *starting_bdy*: Decimal number 3-31 or register (2) - (12).
,KEY=*key number*	*key number*: Decimal number 0-15 or register (2) - (12). **Note**: KEY is valid only when you also specify SP. You cannot specify both KEY and CALLRKY=YES.
,CALLRKY=NO ,CALLRKY=YES	**Default**: CALLRKY=NO **Note**: You cannot specify both CALLRKY=YES and KEY.
,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.

,RTCD=*rtcd addr*	*rtcd addr*: RX-type address, register (15), or register (2) - (12). **Default**: RTCD=(15).
,COND=YES ,COND=NO	**Default**: COND=NO
,CHECKZERO=YES ,CHECKZERO=NO	**Default**: CHECKZERO=NO
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

OBTAIN
Requests that the system obtain virtual storage.

,LENGTH=*length value*
,LENGTH=(*max amount,min amount***)**
Specifies the amount of storage the system is to obtain. *length value* specifies the length, in bytes, of the requested virtual storage. *max length* and *min length* specify the maximum and minimum amounts of storage. These numbers should be a multiple of 8; if they are not, the system uses the next higher multiple of 8.

If you specify LENGTH=(*max amount,min amount*), the system returns a value in general purpose register 0 to tell you the amount of storage it obtained.

,ADDR=*stor addr*
Specifies the location where the system returns the address of the storage it allocates.

,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

,SP=*subpool number*
Specifies the subpool number for the storage. Valid subpools for programs in problem state are 0 - 127, 131, and 132. See the discussion of subpool handling in *z/OS MVS Programming: Assembler Services Guide* for information and requirements pertaining to specific subpools. If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. If you omit this parameter, the system uses subpool 0.

,BNDRY=DBLWD
,BNDRY=PAGE
Specifies whether the storage is to be aligned on a doubleword boundary (DBLWD) or a page boundary (PAGE). The default is BNDRY=DBLWD.

,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. Do not specify CONTBDY on a variable-length request.

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. Do not specify STARTBDY on a variable-length request.

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).

,KEY=*key number*
Indicates the storage key of the storage to be obtained. You may obtain storage in your storage key or in key 9. If you pass the storage key in a register, it must be in bits 24-27 in that register. KEY is valid only when SP is specified, and applies to subpools 131 and 132 only. See the discussion of subpool handling in *z/OS MVS Programming: Assembler Services Guide* for information on system-assigned defaults and authorization requirements pertaining to specific subpools.

,CALLRKY=NO
,CALLRKY=YES
Specifies how the system assigns the key for the storage to be obtained:

CALLRKY=NO
The system assigns the value according to the specified subpool:

- For subpools 131 and 132, the system assigns the value specified on the KEY parameter (or 0, if the KEY parameter is omitted) as the storage key
- For subpools 0-127, the system assigns the value from the TCB key at the time of the first request to obtain storage. See the discussion of subpool handling in *z/OS MVS Programming: Assembler Services Guide* for information on system-assigned defaults and authorization requirements pertaining to specific subpools.

CALLRKY=YES
The system assigns the caller's current PSW key as the storage key. When you specify CALLRKY=YES, do not also specify KEY. Specify CALLRKY only when obtaining storage from subpools 131 and 132. For all other subpools, the system ignores the CALLRKY parameter.

The default is CALLRKY=NO.

,LOC=24
,LOC=(24,31)
,LOC=(24,64)
,LOC=31
,LOC=(31,31)
,LOC=(31,64)
,LOC=RES
,LOC=(RES,31)
,LOC=(RES,64)
,LOC=EXPLICIT
,LOC=(EXPLICIT,24)
,LOC=(EXPLICIT,31)
,LOC=(EXPLICIT,64)
Specifies the location of virtual storage and central (also called real) storage. This is especially helpful for callers with 24-bit dependencies. When LOC is specified, central storage is allocated anywhere until the storage is fixed (for example, using the PGSER macro). You can specify the location of central storage (after the storage is fixed) and virtual storage (whether or not the storage is fixed) using the following LOC parameter values:

LOC=24 indicates that central and virtual storage are to be located below 16 megabytes. LOC=24 must not be used to allocate disabled reference (DREF) storage.

Note: Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW is still supported, but IBM recommends using LOC=24 instead.

LOC=(24,31) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=(BELOW,ANY) is the same as specifying LOC=(24,31). LOC=(BELOW,ANY) is still supported, but IBM recommends using LOC=(24,31) instead.

LOC=(24,64) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere in 64-bit storage.

LOC=31 and LOC=(31,31) indicate that virtual and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC =31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64) indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

When you use LOC=RES to allocate storage that can reside either above or below 16 megabytes, LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be located below 16 megabytes. If the caller resides above 16 megabytes, virtual and central storage are to be located either above or below 16 megabytes.

LOC=(RES,31) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere below 2 gigabytes. In either case, central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=(RES,ANY) is the same as specifying LOC=(RES,31). LOC=(RES,ANY) is still supported, but IBM recommends using LOC=(RES,31) instead.

LOC=(RES,64) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere in 31-bit storage. In either case, central storage can be located anywhere in 64-bit storage.

Note: If your program resides below 16 megabytes but runs with 31-bit addressing mode, you can specify LOC=RES (as a default or explicitly) or LOC=(RES,31) to obtain storage from a subpool supported only above 16 megabytes. Do not specify subpools supported only above 16 megabytes on requests using LOC=RES or LOC=(RES,31) if your program resides below 16 megabytes and runs with 24-bit addressing.

LOC=EXPLICIT, LOC=(EXPLICIT,24), LOC=(EXPLICIT,31), or LOC=(EXPLICIT,64) specify that the requested virtual storage is to be located at the address specified with the INADDR parameter, which is required with EXPLICIT. EXPLICIT is valid only for subpools 0-127, 131, and 132. You cannot specify the BNDRY or LENGTH=(*max amount,min amount*) parameter with EXPLICIT.

Note: Specifying LOC=(EXPLICIT,BELOW) is the same as specifying LOC=(EXPLICIT,24). Specifying LOC=(EXPLICIT,ANY is the same as specifying LOC=(EXPLICIT,31). The older specifications are still supported, but IBM recommends using the newer specifications instead.

LOC=(EXPLICIT,31) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere below 2 gigabytes.

LOC=(EXPLICIT,24) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage is to be located below 16 megabytes. The virtual storage address specified on the INADDR parameter must be below 16 megabytes.

LOC=EXPLICIT and LOC=(EXPLICIT,64) indicate that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere in 64-bit storage.

When you specify EXPLICIT on a request for storage from the same virtual page as previously requested storage, you must request it in the same key, subpool, and central storage area as on the previous storage request. For example, if you request virtual storage backed with central storage below 16 megabytes, any subsequent requests for storage from that virtual page must be specified as LOC=(EXPLICIT,24).

,RTCD=*rtcd addr*
Specifies the location where the system is to store the return code. This parameter is valid only with COND=YES. The return code is also in GPR 15.

,COND=NO
,COND=YES
COND=YES specifies that the active unit of work should not be abnormally terminated if there is insufficient contiguous virtual storage to satisfy the request, and instead should return to the caller with a non-zero return code. Use of COND=YES 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. If you specify COND=YES, you may also specify the RTCD parameter to define the location where the system is to store the return code.

COND=NO indicates that the request is unconditional. The system abnormally terminates the active unit of work if the STORAGE OBTAIN request cannot complete successfully. This situation occurs if the parameters passed on the request are incorrect or inconsistent, if the system encounters internal errors, or if there is not enough contiguous virtual storage to satisfy the request. COND=NO is the default.

,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 no 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 STORAGE 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 macro 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

STORAGE OBTAIN might issue the hexadecimal abend codes in the following list. For detailed abend code information, see *z/OS MVS System Codes*.

178	278	378	478	778
878	978	A78	B78	D78

Return and Reason Codes

When control returns from the STORAGE OBTAIN request and you specified a conditional request, GPR 15 (and *rtcd addr*, if you coded RTCD) contains one of the following hexadecimal return codes:

Table 33. Return Codes for STORAGE OBTAIN

Return Code	Meaning and Action
0	**Meaning**: Successful completion. **Action**: None.
4	**If you did not specify EXPLICIT on the LOC parameter**: **Meaning**: Environmental error. Virtual storage was not obtained because insufficient storage is available. **Action**: Consult the system programmer to see if you have exceeded an installation-determined private storage limit. **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 your region size is too small, consult the system programmer about increasing the region size.
8	**Meaning**: System error. Virtual storage was not obtained because the system has insufficient central storage to back the request. **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected.
C	**Meaning**: System error. Virtual storage was not obtained because the system cannot page in the page table associated with the storage to be allocated. **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected.
10	**Meaning**: Program error. Virtual storage was not obtained for one of the reasons listed below. This reason code applies only to STORAGE 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.

RELEASE Option of STORAGE

The STORAGE macro with the RELEASE parameter requests that the system release an area of virtual storage or an entire virtual storage subpool, previously allocated through the STORAGE or GETMAIN macro. The system abends the active task if the specified virtual storage does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not allocated to the task identified as the owning task.

Input Register Information

Before issuing the STORAGE macro with the RELEASE parameter, 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	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

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.

Syntax

The STORAGE macro with the RELEASE option is written as follows:

---

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

ƀ — One or more blanks must precede STORAGE.

STORAGE

ƀ — One or more blanks must follow STORAGE.

RELEASE

,LENGTH=*length value*,ADDR=*stor addr*

,LENGTH=*length value*,ADDR=*stor addr*,SP=*subpool number*	*length value*: Symbol, decimal number, or register (0), (2) - (12). *stor addr*: RX-type address or register (1) - (12). *subpool number*: Symbol, decimal number 0-127, 131, 132, or register (2) - (12), (15). **Default**: SP=0.
,KEY=*key number*	*key number*: Decimal number 0-15 or register (2) - (1 2). **Note**: KEY is valid only when SP is specified.
,RTCD=*rtcd addr*	*rtcd addr*: RX-type address, register (15), or register (2) - (12). **Default**: RTCD=(15).
,COND=YES ,COND=NO	**Default**: COND=NO
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

RELEASE
Requests that the system release virtual storage.

,LENGTH=*length value*
Specifies the number of bytes of storage that the system is to release. If you specify LENGTH, you must also specify ADDR. To free an entire subpool, use SP instead of LENGTH and ADDR. Do not specify a length value of 0 with an address of 0. This combination causes STORAGE RELEASE to free the subpool specified with the SP parameter, or subpool 0 if the SP parameter is omitted.

,ADDR=*stor addr*
Specifies the address of the storage to be released. If you specify ADDR, you must also specify LENGTH. To free an entire subpool, use SP instead of LENGTH and ADDR.

,SP=*subpool number*
Specifies the subpool number for the storage to be released. The valid subpool numbers are 0-127, 131, and 132. If you specify the subpool in a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. If you omit this parameter, the system uses subpool 0.

A request to release all the storage in a subpool is known as a **subpool release**. To issue a subpool release, use SP to indicate the subpool and do not specify LENGTH or ADDR. A caller in problem state can issue a subpool release for subpools 1-127, 131, and 132. A caller in problem state cannot issue a subpool release for subpool 0. See the description of subpool handling in *z/OS MVS Programming: Assembler Services Guide* for information and requirements pertaining to specific subpools.

,KEY=*key number*
Indicates the storage key of the storage to be released. The valid storage keys are your program's storage key or key 9. If you pass the storage key in a register, it must be in bits 24-27 in that register. KEY is valid only when SP is

specified and applies only to subpools 131 and 132. KEY allows you to release storage in the specified storage key. See the discussion of subpool handling in *z/OS MVS Programming: Assembler Services Guide* for information on authorization requirements pertaining to specific subpools.

,RTCD=*rtcd addr*
Specifies the location where the system is to store the return code. This parameter is valid only for conditional requests. The return code is also in GPR 15.

,COND=NO
,COND=YES
Specifies whether the request is unconditional or conditional.

COND=YES specifies that the task should not abend if the system cannot release the storage. However, the system cannot prevent some abends. The RTCD parameter specifies the location where the system is to store a return code.

COND=NO specifies that the system is to abend the active task if it cannot release the storage. COND=NO is the default.

,RELATED=*value*
Specifies information used to self-document macro 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

STORAGE RELEASE might issue the hexadecimal abend codes in the following list. For detailed abend code information, see *z/OS MVS System Codes*.

178	278	378	478	778
878	978	A78	B78	D78

Return and Reason Codes

When the STORAGE macro returns control to your program and you specified a conditional request, GPR 15 (and *rtcd addr*, if you coded RTCD) contains one of the following hexadecimal return codes:

Table 34. Return Codes for the STORAGE RELEASE

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.

Table 34. Return Codes for the STORAGE RELEASE (continued)

Return Code	Meaning and Action
8	**Meaning**: Program error. No virtual storage was freed because part of the storage area to be freed is fixed. **Action**: Check for the following kinds of errors: • You passed an incorrect storage area address to the STORAGE macro. • You attempted to free storage that is fixed.

Examples of the OBTAIN and RELEASE Options

Example 1

Request that the system obtain 1000 bytes of virtual storage from subpool 127 and return its address in register 3. If the request fails, the system is to abnormally end the caller.

```
         LA      2,1000
         STORAGE OBTAIN,LENGTH=(2),ADDR=(3),SP=127,LOC=ANY,COND=NO
         .
* Release 1000 bytes from subpool 127 and abnormally end the
* caller if the request fails. Assume that the length of the storage
* is still in register 2 and the address of the storage is still in
* register 3.
         .
  STORAGE RELEASE,LENGTH=(2),ADDR=(3),SP=127,COND=NO
         .
```

Example 2

Request that the system obtain 4096 bytes from subpool 101 and return the address at the location defined by the RX-type address STRGA. If the request fails, the system is to save a return code at MY_RC.

```
         STORAGE OBTAIN,LENGTH=ONE_PAGE,ADDR=STRGA,SP=MY_SUBPOOL,     X
               LOC=ANY,COND=YES,RTCD=MY_RC
         .
* Release 4096 bytes from subpool 101.
         .
         STORAGE RELEASE,LENGTH=ONE_PAGE,ADDR=STRGA,SP=MY_SUBPOOL,    X
               COND=YES,RTCD=MY_RC
         .
MY_RC    DS F
STRGA    DS F
ONE_PAGE EQU   4096
MY_SUBPOOL EQU   101
```

Example 3

Request that the system obtain 4096 bytes from subpool 101. If that much is not available, settle for a minimum of 1024 bytes. The system is to return the address of the storage at the RX-type address STRGA. If the request fails, the system is to store a return code at MY_RC.

```
         STORAGE OBTAIN,LENGTH=(ONE_PAGE,ONE_K),ADDR=STRGA,       X
               SP=MY_SUBPOOL,LOC=ANY,COND=YES,RTCD=MY_RC
         ST      0,STRG_LEN
         .
* Release the storage in subpool 101. The address of the
* storage is at the RX-type address 'STRGA'.  Note that
* LENGTH=STRG_LEN is not valid.
         .
         L       3,STRG_LEN
         STORAGE RELEASE,LENGTH=(3),ADDR=STRGA,SP=MY_SUBPOOL,      X
```

```
               COND=YES,RTCD=MY_RC
         .
MY_RC    DS F
STRG_LEN DS F
STRGA    DS F
ONE_PAGE EQU   4096
ONE_K    EQU   1024
MY_SUBPOOL EQU   101
```

Example 4

Code the instructions to set up an 18-word save area, such as one that a program in AR address space control (ASC) mode would obtain to call a program in primary mode. The program issuing the STORAGE macro is in 31-bit addressing mode, and the code is reentrant.

```
PGM   CSECT
PGM   AMODE 31
PGM   RMODE ANY
      BAKR   14,0               SAVE CALLER'S ARS, GPRS AND RETURN
*                               ADDRESS ON LINKAGE STACK
      SAC    512                SWITCH TO AR ASC MODE
      LAE    12,0(15,0)         SET UP PROGRAM BASE REGISTER AND AR
      USING  PGM,12
      STORAGE OBTAIN,LENGTH=72  GET REENTRANT SAVEAREA
      LAE  13,0(1,0)            PUT SAVEAREA ADDRESS IN AR/GPR 13
      MVC   4(4,13),=C'F1SA'    PUT ACRONYM INTO SAVEAREA TO
*                               INDICATE STATUS SAVED ON LINKAGE STACK
      .
*   BEGIN PROGRAM CODE HERE
```

To release this save area, issue the following instructions:

```
      .
      LAE  1,0(13,0)           COPY SAVEAREA ADDRESS
      STORAGE  RELEASE,ADDR=(1),LENGTH=72  FREE SAVEAREA
      .
      SLR  15,15       SET RETURN CODE OF ZERO
      PR               RETURN TO CALLER, RESTORE CALLERS STATUS
```

SYMRBLD — Building a Symptom Record

Description

The SYMRBLD macro generates code to build a symptom record. A symptom record is a data area that contains a description of a program failure combined with a description of the environment where the failure occurred. The symptom record consists of six sections. These sections are numbered 1 through 5, including an additional section that is numbered 2.1. The purpose of each section is as follows:

- Section 1 (Environmental Data) - This section is filled in by the SYMREC macro. The environmental data the SYMREC macro stores in this section includes the processor model and serial numbers, data and time, name of the customer installation, and the product ID of the control program.
- Section 2 (Control Data) - This section contains the lengths and offsets of the remaining sections.
- Section 2.1 (Component Data) - This section identifies the application in which the error occurred.
- Section 3 (Primary SDB symptoms) - This section contains the primary string of problem symptoms. This data is used for duplicate problem recognition.
- Section 4 (Secondary SDB symptoms) - This section contains any additional diagnostic values saved at the time of the error.
- Section 5 (Variable Data) - This section contains diagnostic data, such as portions of data areas or parameter lists pertinent to the error.

Input to the SYMRBLD macro is a storage area for the symptom record, and the diagnostic data for sections 2.1, 3, 4, and 5 of the symptom record. The SYMRBLD macro must be invoked several times to build a complete symptom record. The following describes the sequence:

1. Invoke SYMRBLD with the INITIAL parameter to initialize sections 1 and 2, and provide application data for section 2.1.
2. Invoke SYMRBLD with the PRIMARY parameter to store symptoms into section 3. You may invoke this parameter more than once for one error.
3. Optionally invoke SYMRBLD with the SECONDARY parameter to store symptoms into section 4.
4. Optionally invoke SYMRBLD with the VARIABLE parameter to store data into section 5.
5. Invoke SYMRBLD with the COMPLETE parameter to set the lengths of sections 3, 4, and 5 in section 2.1 and optionally code SYMRBLD to invoke the SYMREC macro for recording to the logrec data set. If you do not code SYMRBLD to invoke the SYMREC macro, your records will not be recorded to the logrec data set.
6. Invoke SYMRBLD with the RESET parameter to rebuild the symptom record using the same storage area and application information that was specified using the INITIAL parameter. The RESET parameter is useful when the primary, secondary, and variable sections of the symptom record are to be changed but the application information in section 2.1 remains the same.

The following description of the SYMRBLD macro is divided into six sections:

- SYMRBLD with the INITIAL parameter
- SYMRBLD with the PRIMARY parameter
- SYMRBLD with the SECONDARY parameter
- SYMRBLD with the VARIABLE parameter

© Copyright IBM Corp. 1988, 2002

- SYMRBLD with the COMPLETE parameter
- SYMRBLD with the RESET parameter

There is no list or execute form of the macro.

Environment

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, secondary, or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks 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 caller's dispatchable unit access list (DU-AL).

Programming Requirements

The maximum size of the symptom record is 1900 bytes. In addition to providing storage for the symptom record, 100 bytes must be provided for a work area; therefore, the maximum amount of storage needed is 2000 bytes.

The symptom record storage must reside in the primary address space.

Restrictions

None.

Input Register Information

When specifying SYMRBLD COMPLETE with INVOKE=YES (the default) the caller must ensure that register 13 points to a standard 72-byte save area.

Once you specify SR on SYMRBLD INITIAL and you plan to specify either SYMRBLD PRIMARY, SYMRBLD SECONDARY, SYMRBLD VARIABLE, or SYMRBLD COMPLETE without respecifying the SR parameter, you must put the address of the storage area into register 1.

Output Register Information

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

Register	Contents
0	Reason code from the SYMREC macro if you code SYMRBLD COMPLETE with INVOKE=YES; 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 from the SYMREC macro if you code SYMRBLD COMPLETE with INVOKE=YES; 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 SYMRBLD macro with the INITIAL option is written as follows:

---

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

INITIAL	
,SR=*storage addr*	*storage addr*: RX-type address or address in register (2)-(12).
,PRIMLEN=*primary length*	*primary length*: Decimal digit, RX-type address, or address in register (2)-(12).
,SECLEN=*secondary length*	*secondary length*: Decimal digit, RX-type address, or address in register (2)-(12). **Default**: 0
,VARLEN=*variable length*	*variable length*: Decimal digit, RX-type address, or address in register (2)-(12). **Default**: 0
,ARCHLEV=10	This is the architecture level of the symptom record.
,COMPDSC=*comp desc*	*comp desc*: RX-type address or address in register (2)-(12).
,PROBLEM=*problem id*	*problem id*: RX-type address or address in register (2)-(12).
,SERVLEV=*service level*	*service level*: RX-type address or address in register (2)-(12).
,NOCONVERTS	
,PROGRAM=*progname*	*progname*: RX-type address or address in register (2)-(12).

,PROGLEV=*proglevel* *proglevel*: RX-type address or address in register (2)-(12).

Parameters

The parameters for SYMRBLD INITIAL are explained as follows:

INITIAL
Sets sections 1, 2, and 2.1 of the symptom record to zero, and initializes the offsets of sections 3, 4, and 5 in section 2.1.

,SR=*storage addr*
Specifies the address of the storage area, on a doubleword boundary, used for the symptom record. The storage area must reside in the primary address space.

The maximum size of the symptom record is 1900 bytes. Sections 1, 2, and 2.1 use 212 bytes of the total 1900 bytes. Sections 3, 4, and 5 use the remaining 1688 bytes. In addition to providing storage for the symptom record, 100 bytes must be provided for a work area, therefore, the maximum amount of storage needed is 2000 bytes.

Use the PRIMLEN, SECLEN, and VARLEN parameters to specify the length of sections 3, 4, and 5, respectively.

,PRIMLEN=*primary length*
Specifies the address of a required halfword input variable that contains the maximum length in bytes of the primary symptom string. You can also directly specify a decimal digit for the length (for example, PRIMLEN=900). If you use register notation, the register contains the address of the length rather than the length itself.

The following formula calculates the length of the primary symptom string:

```
Lengths of all SDBKEYs + length of all data provided with
the DATA keyword + the number of times SDBKEY is specified
+ the length of all data specified with the SDBSTRING keyword
+ the number of times the SDBSTRING keyword is provided.
```

Note that this field cannot be zero and the maximum size of the entire symptom record is 1900 bytes.

,SECLEN=*secondary length*
Specifies the address of an optional halfword input variable that contains the maximum length in bytes of the secondary symptom string. You can also directly specify a decimal digit for the length (for example, SECLEN=900). If you use register notation, the register contains the address of the length rather than the length itself.

The following formula calculates the length of the secondary symptom string:

```
Lengths of all SDBKEYs + length of all data provided with
the DATA keyword + the number of times SDBKEY is specified
+ the length of all data specified with the SDBSTRING keyword
+ the number of times the SDBSTRING keyword is provided.
```

Note that the maximum size of the entire symptom record is 1900 bytes.

If a length of zero is specified, the secondary symptom string is ignored. If SECLEN is not specified, the default is zero.

,VARLEN=*variable length*
Specifies the address of an optional halfword input variable that contains the maximum length in bytes of the variable data section. You can also directly specify a decimal digit for the length (for example, VARLEN=900). If you use register notation, the register contains the address of the length rather than the length itself.

The following formula calculates the length of the variable data section:

```
The length provided must be the total length of the variable data items
+ the number of items (x) 4.
```

(The 4 is for the 2 byte key + 2 bytes for the length.) Note that the maximum size of the entire symptom record is 1900 bytes.

If a length of zero is specified, section 5 is ignored. If VARLEN is not specified, the default is zero.

,ARCHLEV=10
Specifies the architecture level of the symptom record. The only valid value is 10.

,COMPDSC=*comp desc*
Specifies the address of an optional 32-character input text description of the failing module's subfunction; for example, IOS - IOSB Analysis Routine.

,PROBLEM=*problem id*
Specifies the address of an optional 8-character input problem identifier used to associate the symptom record with other symptom records or with other problem indicators.

,SERVLEV=*service level*
Specifies the address of an optional 8-character input service level. When a value is provided, the code is normally at a higher level than the release level. The values of this field can be any information that is indicative of the service level; for example, PTF#, APAR#, or user modification number.

,NOCONVERTS
Indicates no data conversion from hexadecimal to EBCDIC is needed for this symptom record.

,PROGRAM=*progname*
Specifies the address of a required 8-character input variable that contains the name of the failing program. When this parameter is specified, the PIDS/aaaaaaaa SDB symptom is automatically put into section 3 of the symptom record. **aaaaaaaa** indicates the *progname*.

,PROGLEV=*proglevel*
Specifies the address of a required 8-character input variable that contains the name of the program major level.

Syntax

The standard form of the SYMRBLD macro with the PRIMARY option is written as follows:

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

PRIMARY

,SR=*storage addr*	*storage addr*: RX-type address or address in register (2)-(12).
,SDBSTRING=*SDB string* ,SDBKEY=*SDB key*	*SDB string*: RX-type address or address in register (2)-(12). *SDB key*: SDB key name, or SDB key literal in single quotes. See the parameter description for a list of valid SDB key names and literals. **Note:** You must code either SDBSTRING or SDBKEY or both.
,SDBLEN=*SDB length* ,SDBLENVAR=*SDB variable*	*SDB length*: Decimal digit 1-256, or register (2)-(12). *SDB variable*: RX-type address or address in register (2)-(12). **Notes:** 1. If you use register notation for *SDB length*, the register contains the length itself rather than the address of the length. 2. SDBLEN (or SDBLENVAR) is valid with SDBSTRING only.
,DATA=*data*	*data*: RX-type address or address in register (2)-(12). **Note:** DATA is required with SDBKEY only.
,LEN=*data length* ,LENVAR=*data variable*	*data length*: Decimal digit 1-13, or register (2)-(12). *data variable*: RX-type address or address in register (2)-(12). **Notes:** 1. If you use register notation for *data length*, the register contains the length itself rather than the address of the length. 2. LEN (or LENVAR) is valid with DATA only.
,CONVERT=YES ,CONVERT=NO	**Default**: CONVERT=NO **Note**: CONVERT is valid with DATA only.
,TYPE=TEST ,TYPE=NOTEST	**Default**: TYPE=TEST

Parameters

The parameters for SYMRBLD PRIMARY are explained as follows:

PRIMARY

Indicates that the symptom data provided is concatenated to section 3, the primary symptom string. The primary symptom string is an EBCDIC character string of problem symptoms. The primary symptom string is used to eliminate reporting duplicate problems repeatedly.

You would use the primary symptom string because, in most cases, the PIDS/aaaaaaaa symptom is in section 3 of the symptom record. When the symptom record is initialized by invoking SYMRBLD INITIAL, the symptom is created from the data supplied with the PROGRAM parameter and is placed as the first symptom in section 3.

The suggested minimum list of symptoms includes:
- Return or reason codes - PRCS/aaaaaaaa
- CSECT name - RIDS/aaaaaaaa
- Load module name - RIDS/aaaaaaaa#L

Note: The following restrictions apply to symptoms in the primary symptom string:
- The symptom data cannot contain imbedded blanks. The '#' is used to substitute for desired blanks.
- The total length of each symptom may not exceed 15 characters. The symptom length includes the SDB key, a slash, and the EBCDIC data. Remember that hexadecimal data doubles in length when converted to EBCDIC.

,SR=*storage addr*

Specifies the address of the storage area, on a doubleword boundary, used for the symptom record. This is the same storage area you specified on SYMRBLD INITIAL. If you do not specify SR with SYMRBLD PRIMARY, the default is to use the storage area address you placed in register 1.

,SDBSTRING=*SDB string*

Specifies the address of an optional character input string to be added to the primary symptom string. The data is a list of symptoms separated by a blank. A symptom is an SDB key followed by a slash and EBCDIC data.

You must code either SDBSTRING or SDBKEY or both. When you code both on the same macro, the data provided with the SDBSTRING parameter is put into the symptom string first.

,SDBKEY=*SDB key*

Specifies an optional name from the set of SDB keys. You can provide the SDB key name, or specify the SDB key literal in single quotes (for example, specify either SDBKEY=SDBAB_S, or SDBKEY='AB/S').

You must code either SDBSTRING or SDBKEY or both. When you code both on the same macro, the data provided with the SDBSTRING parameter is put into the symptom string first.

The following table contains the valid SDB key names and literals:

Table 35. Valid SDB Key Names and Literals

SDB Key Name	SDB Key Literal	Description
SDBAB_S	AB/S	System abend or program check.
SDBAB_U	AB/U	User abend code.
SDBADRS	ADRS/	Any software routine, CSECT, or program address; displacement within a routine; or offset within a field or data area.
SDBDEVS	DEVS/	IBM device types.

Table 35. Valid SDB Key Names and Literals (continued)

SDB Key Name	SDB Key Literal	Description
SDBFLDS	FLDS/	A field, data area, or label involved with the problem. If a field name is longer than 10 characters, use two keys and split the name of the field.
SDBLVLS	LVLS/	The system release or program product/component level where the problem occurs.
SDBMS	MS/	Program- or device-issued message. If there is no identifier, enter the message as it appears and MS/NOID to denote this.
SDBOPCS	OPCS/	Software program operation code, I/O read/write command codes, teleprocessing operation codes and request codes.
SDBOVS	OVS/	Overlaid storage.
SDBPCSS	PCSS/	Any software statement, JCL, operator or user commands, parameters, program language statements, data set names, library names, teleprocessing logical and physical unit names, program function keys or other operator keys, environments, process names, procedures or other symptoms which do not fit other key descriptions in this table.
SDBPIDS	PIDS/	Product identifier.
SDBPRCS	PRCS/	Any program-generated return, reason, step, condition, or device status code.
SDBPTFS	PTFS/	Program temporary fix (PTF) or Authorized Program Analysis Report (APAR) associated with the problem.
SDBPUBS	PUBS/	Publication identifier.
SDBREGS	REGS/	A register number associated with the problem, followed by the offset from the PSW.
SDBREGS_CR	REGS/CR	A control register associated with the problem. This symptom is followed with a symptom containing the value in the register.
SDBREGS_FP	REGS/FP	A floating point register associated with the problem. This symptom is followed with a symptom containing the value in the register.
SDBREGS_GR	REGS/GR	A general purpose register associated with the problem. This symptom is followed with a symptom containing the value in the register.
SDBREGS_AR	REGS/AR	An access register associated with the problem. This symptom is followed with a symptom containing the value in the register.
SDBRIDS	RIDS/	Module CSECT name.
SDBRIDSL	RIDS/	Load module name.
SDBRIDSR	RIDS/	Recovery routine CSECT name.
SDBSIG	SIG/	System- or device-issued operator warning signal.

Table 35. Valid SDB Key Names and Literals (continued)

SDB Key Name	SDB Key Literal	Description
SDBVALU	VALU/	Contents of a register. This SDB keyword must be preceded with one of the following: REGS/CRhh, REGS/FPhh, or REGS/GRhh.
SDBVALU_B	VALU/B	Binary value of a field in error. This SDB key must be preceded by the name of the field. The most appropriate SDB key is FLDS/.
SDBVALU_C	VALU/C	Character value of a field in error. This SDB key must be preceded by the name of the field. The most appropriate SDB key is FLDS/.
SDBVALU_H	VALU/H	Hexadecimal value of a field in error. This SDB key must be preceded by the name of the field. The most appropriate SDB key is FLDS/.
SDBWS_D	WS/D	System- or device-issued disabled WAIT code.
SDBWS_E	WS/E	System- or device-issued enabled WAIT code.

,SDBLEN=*SDB length*
Specifies an optional decimal value from 1 to 256 that is the length of the data provided. If you use register notation, the register contains the length itself rather than the address of the length. This parameter is mutually exclusive with the SDBLENVAR parameter, and is valid with SDBSTRING only.

,SDBLENVAR=*SDB variable*
Specifies the address of an optional halfword that contains the length of the data provided. The length of the data must be from 1 to 256 bytes. This parameter is mutually exclusive with the SDBLEN parameter, and is valid with SDBSTRING only.

,DATA=*data*
Specifies the address of the area that contains the data associated with the key specified by the SDBKEY parameter. DATA is required with SDBKEY only.

,LEN=*data length*
Specifies an optional decimal value from 1 to 13 that is the length of the data provided. If you use register notation, the register contains the length itself rather than the address of the length. This parameter is mutually exclusive with the LENVAR parameter, and is valid with DATA only.

,LENVAR=*data variable*
Specifies the address of an optional halfword that contains the length of the data provided. The length of the data must be from 1 to 13 bytes. This parameter is mutually exclusive with the LEN parameter, and is valid with DATA only.

,CONVERT=YES
,CONVERT=NO
Indicates that 1 to 4 bytes of hexadecimal data specified by the DATA parameter should be converted to EBCDIC. If the length of the hexadecimal data is greater than 4 bytes, the results of the conversion are unpredictable.

If CONVERT is specified with the user abend code SDB key, SDBAB_U, the hexadecimal data is converted to decimal EBCDIC.

The default is CONVERT=NO. CONVERT is valid with DATA only.

,TYPE=TEST

,TYPE=NOTEST
Specifies whether code is generated to test if the data fits in the symptom record before storing the data. TYPE=NOTEST indicates that the data and key are unconditionally moved into the symptom record.

The default is TYPE=TEST.

Syntax

The standard form of the SYMRBLD macro with the SECONDARY option is written as follows:

---

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

SECONDARY

,SR=*storage addr*	*storage addr*: RX-type address or address in register (2)-(12).
,SDBSTRING=*SDB string*	*SDB string*: RX-type address or address in register (2)-(12).
,SDBKEY=*SDB key*	*SDB key*: SDB key name, or SDB key literal in single quotes. See the parameter description for a list of valid SDB key names and literals. **Note**: You must code either SDBSTRING or SDBKEY or both.
,SDBLEN=*SDB length*	*SDB length*: Decimal digit 1-256, or register (2)-(12).
,SDBLENVAR=*SDB variable*	*SDB variable*: RX-type address or address in register (2)-(12). **Notes:** 1. If you use register notation for *SDB length*, the register contains the length itself rather than the address of the length. 2. SDBLEN (or SDBLENVAR) is valid with SDBSTRING only.
,DATA=*data*	*data*: RX-type address or address in register (2)-(12). **Note**: DATA is required with SDBKEY only.
,LEN=*data length*	*data length*: Decimal digit 1-13, or register (2)-(12).
,LENVAR=*data variable*	*data variable*: RX-type address or address in register (2)-(12). **Notes:** 1. If you use register notation for *data length*, the register contains the length itself rather than the address of the length. 2. LEN (or LENVAR) is valid with DATA only.
,CONVERT=YES ,CONVERT=NO	**Default**: CONVERT=NO **Note**: CONVERT is valid with DATA only.
,TYPE=TEST ,TYPE=NOTEST	**Default**: TYPE=TEST

Parameters

The parameters for SYMRBLD SECONDARY are explained as follows:

SECONDARY

Indicates that the symptom data provided is concatenated to section 4, the secondary symptom string. The secondary symptom string is an EBCDIC character string of problem symptoms, SDB key/data pairs. The purpose of the secondary symptom string is to save diagnostic data at the time of the error. This data may not be duplicated for each instance of the problem.

The suggested minimum list of symptoms includes:

- Module assembly level - LVLS/aaa
- Field name related to the error and contents - FLDS/av10 VALU/Cav8

  Binary and hex data can be provided with the VALU/B and VALU/H keys.

Note: The following restrictions apply to symptoms in the secondary symptom string:

- The symptom data cannot contain imbedded blanks. The '#' is used to substitute for desired blanks.
- The total length of each symptom (key/data) may not exceed 15 characters. The symptom length includes the SDB key, a slash, and the EBCDIC data. Remember that hexadecimal data doubles in length when converted to EBCDIC.

,SR=*storage addr*

Specifies the address of the storage area, on a doubleword boundary, used for the symptom record. This is the same storage area you specified on SYMRBLD INITIAL. If you do not specify SR with SYMRBLD SECONDARY, the default is to use the storage area address you placed in register 1.

,SDBSTRING=*SDB string*

Specifies the address of an optional character input string to be added to the secondary symptom string. The data is a list of symptoms separated by a blank. A symptom is an SDB key followed by a slash and EBCDIC data.

You must code either SDBSTRING or SDBKEY or both. When you code both on the same macro, the data provided with the SDBSTRING parameter is put into the symptom string first.

,SDBKEY=*SDB key*

Specifies an optional name from the set of SDB keys. You can provide the SDB key name, or specify the SDB key literal in single quotes (for example, specify either SDBKEY=SDBAB_S, or SDBKEY='AB/S'). See Table 35 on page 611 for valid SDB key names and literals.

You must code either SDBSTRING or SDBKEY or both. When you code both on the same macro, the data provided with the SDBSTRING parameter is put into the symptom string first.

,SDBLEN=*SDB length*

Specifies an optional decimal value from 1 to 256 that is the length of the data provided. If you use register notation, the register contains the length itself rather than the address of the length. This parameter is mutually exclusive with the SDBLENVAR parameter, and is valid with SDBSTRING only.

,SDBLENVAR=*SDB variable*
Specifies the address of an optional halfword that contains the length of the data provided. The length of the data must be from 1 to 256 bytes. This parameter is mutually exclusive with the SDBLEN parameter, and is valid with SDBSTRING only.

,DATA=*data*
Specifies the address of the area that contains the data associated with the key specified by the SDBKEY parameter. DATA is required with SDBKEY only.

,LEN=*data length*
Specifies an optional decimal value from 1 to 13 that is the length of the data provided. If you use register notation, the register contains the length itself rather than the address of the length. This parameter is mutually exclusive with the LENVAR parameter, and is valid with DATA only.

,LENVAR=*data variable*
Specifies the address of an optional halfword that contains the length of the data provided. The length of the data must be from 1 to 13 bytes. This parameter is mutually exclusive with the LEN parameter, and is valid with DATA only.

,CONVERT=YES
,CONVERT=NO
Indicates that 1 to 4 bytes of hexadecimal data specified by the DATA parameter should be converted to EBCDIC. If the length of the hexadecimal data is greater than 4 bytes, the results of the conversion are unpredictable.

If CONVERT is specified with the user abend code SDB key, SDBAB_U, the hexadecimal data is converted to decimal EBCDIC.

The default is CONVERT=NO. CONVERT is valid with DATA only.

,TYPE=TEST
,TYPE=NOTEST
Specifies whether code is generated to test if the data fits in the symptom record before storing the data. TYPE=NOTEST indicates that the data and key are unconditionally moved into the symptom record.

The default is TYPE=TEST.

Syntax

The standard form of the SYMRBLD macro with the VARIABLE option is written as follows:

---

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

VARIABLE

,SR=*storage addr*	*storage addr*: RX-type address or address in register (2)-(12).
,S5KEY=*5key*	*5key*: Section 5 key name, or section 5 key literal in single quotes. See the parameter description for valid section 5 key names and literals.
,DATA=*data*	*data*: RX-type address or address in register (2)-(12).
,LEN=*data length* ,LENVAR=*data variable*	*data length*: Decimal digit 1-256, or register (2)-(12). *data variable*: RX-type address or address in register (2)-(12). **Note:** If you use register notation for *data length*, the register contains the length itself rather than the address of the length.
,TYPE=NOTEST ,TYPE=TEST	**Default**: TYPE=TEST

Parameters

The parameters for SYMRBLD VARIABLE are explained as follows:

VARIABLE
Indicates that the symptom data provided is concatenated to section 5, the variable data section. The variable data section is in key/length/data format. The purpose of the variable data section is to provide additional serviceability data for debugging. Examples of serviceability data are a parameter list, a text description of the problem, or a portion of a data area.

The VARIABLE parameter must be specified once for each symptom provided in key/length/data format.

,SR=*storage addr*
Specifies the address of the storage area, on a doubleword boundary, used for the symptom record. This is the same storage area you specified on SYMRBLD INITIAL. If you do not specify SR with SYMRBLD VARIABLE, the default is to use the storage area address you placed in register 1.

,S5KEY=*5key*
Specifies the key that describes the data in section 5 of the symptom record. You can provide the section 5 key name, or specify the section 5 key literal in single quotes (for example, specify either S5KEY=S5EBCDIC, or S5KEY=‘F000’).

The following table contains the two valid section 5 key names and literals:

Table 36. Valid Section 5 Key Names and Literals

Section 5 Key Name	Section 5 Key Literal	Description
S5EBCDIC	F000	EBCDIC printable data.
S5HEX	FF00	Hexadecimal data.

,DATA=*data*
Specifies the address of the area that contains the data associated with the key specified by the S5KEY parameter.

,LEN=*data length*
Specifies an optional decimal value from 1 to 256 that is the length of the data

provided. If you use register notation, the register contains the length itself rather than the address of the length. This parameter is mutually exclusive with the LENVAR parameter.

,LENVAR=*data variable*
Specifies the address of an optional halfword that contains the length of the data provided. The length of the data must be from 1 to 256 bytes. This parameter is mutually exclusive with the LEN parameter.

,TYPE=TEST
,TYPE=NOTEST
Specifies whether code is generated to test if the data fits in the symptom record before storing the data. TYPE=NOTEST indicates that the data and key are unconditionally moved into the symptom record.

The default is TYPE=TEST.

Syntax

The standard form of the SYMRBLD macro with the COMPLETE option is written as follows:

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

COMPLETE	
,SR=*storage addr*	*storage addr*: RX-type address or address in register (2)-(12).
,INVOKE=YES ,INVOKE=NO	**Default**: INVOKE=YES
,RETCODE=*return code*	*return code*: RX-type address or address in register (2)-(12). **Note:** RETCODE is valid with INVOKE=YES only.
,RSNCODE=*reason code*	*reason code*: RX-type address or address in register (2)-(12). **Note:** RSNCODE is valid with INVOKE=YES only.

Parameters

The parameters for SYMRBLD COMPLETE are explained as follows:

COMPLETE
Indicates that the symptom record is complete, and is ready to be written to the logrec data set.

SYMRBLD COMPLETE is required before the symptom record can be successfully written to the logrec data set.

,SR=*storage addr*
Specifies the address of the storage area, on a doubleword boundary, used for the symptom record. This is the same storage area you specified on SYMRBLD INITIAL. If you do not specify SR with SYMRBLD COMPLETE, the default is to use the storage area address you placed in register 1.

,INVOKE=NO
,INVOKE=YES
Indicates whether to invoke the SYMREC macro that writes the symptom records out to the logrec data set. For unauthorized programs, your installation controls which programs can write symptom records and whether to write the symptom record to the logrec data set, the job log, both or neither through an installation-written exit. This exit is called ASREXIT. For more information about ASREXIT, see *z/OS MVS Installation Exits*. Records written for authorized programs always go to the logrec data set.

The default is INVOKE=YES.

,RETCODE=*return code*
Specifies the location where the system is to store the return code from the SYMREC macro. (The SYMRBLD macro does not itself generate any return codes.) RETCODE is valid with INVOKE=YES only. The return code is also in general purpose register (GPR) 15 if you code INVOKE=YES.

,RSNCODE=*reason code*
Specifies the location where the system is to store the reason code from the SYMREC macro. (The SYMRBLD macro does not itself generate any reason codes.) RSNCODE is valid with INVOKE=YES only. The reason code is also in GPR 0 if you code INVOKE=YES.

ABEND Codes

None.

Return and Reason Codes (for SYMRBLD COMPLETE,INVOKE=YES)

The SYMRBLD macro itself does not generate any return codes. However, if you specify INVOKE=YES on SYMRBLD COMPLETE (or take the default), you can receive return codes and reason codes from the SYMREC macro. The return code from SYMREC is in GPR 15 (and *return code* if you coded RETCODE); the reason code from SYMREC is in GPR 0 (and *reason code* if you coded RSNCODE). See “Return and Reason Codes” on page 625 for a list of return codes from the SYMREC macro.

Syntax

The standard form of the SYMRBLD macro with the RESET option is written as follows:

---

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

b	One or more blanks must follow SYMRBLD.

RESET

,SR=*storage addr*	*storage addr*: RX-type address or address in register (2)-(12).
,PRIMLEN=*primary length*	*primary length*: Decimal digit, RX-type address, or address in register (2)-(12).
,SECLEN=*secondary length*	*secondary length*: Decimal digit, RX-type address, or address in register (2)-(12).
,VARLEN=*variable length*	*variable length*: Decimal digit, RX-type address, or address in register (2)-(12).

Parameters

The parameters for SYMRBLD RESET are explained as follows:

RESET
Rebuilds the symptom record using the same storage area and application information that was specified using the INITIAL parameter. This is useful when the primary, secondary, and variable sections of the symptom record are to be changed but the application information in section 2.1 remains the same.

,SR=*storage addr*
Specifies the address of the storage area, on a doubleword boundary, used for the symptom record. This is the same storage area you specified on SYMRBLD INITIAL. The storage area must reside in the primary address space.

The maximum size of the symptom record is 1900 bytes. Sections 1, 2, and 2.1 use 212 bytes of the total 1900 bytes. Sections 3, 4, and 5 use the remaining 1688 bytes. In addition to providing storage for the symptom record, 100 bytes must be provided for a work area; therefore, the maximum amount of storage needed is 2000 bytes.

Use the PRIMLEN, SECLEN, and VARLEN parameters to specify the length of sections 3, 4, and 5 respectively.

,PRIMLEN=*primary length*
Specifies the address of an optional halfword input variable that contains the maximum length in bytes of the primary symptom string. You can also directly specify a decimal digit for the length (for example, PRIMLEN=900). If you use register notation, the register contains the address of the length rather than the length itself.

The following formula calculates the length of the primary symptom string:

```
Lengths of all SDBKEYs + length of all data provided with
the DATA keyword + the number of times SDBKEY is specified
+ the length of all data specified with the SDBSTRING keyword
+ the number of times the SDBSTRING keyword is provided.
```

Note that this field cannot be zero and the maximum size of the entire symptom record is 1900 bytes.

If you do not specify PRIMLEN, the length of the primary symptom string will not change from the length you specified on SYMRBLD INITIAL, or on a previous SYMRBLD RESET.

,SECLEN=*secondary length*

Specifies the address of an optional halfword input variable that contains the maximum length in bytes of the secondary symptom string. You can also directly specify a decimal digit for the length (for example, SECLEN=900). If you use register notation, the register contains the address of the length rather than the length itself.

The following formula calculates the length of the secondary symptom string:

```
Lengths of all SDBKEYs + length of all data provided with
the DATA keyword + the number of times SDBKEY is specified
+ the length of all data specified with the SDBSTRING keyword
+ the number of times the SDBSTRING keyword is provided.
```

Note that the maximum size of the entire symptom record is 1900 bytes.

If you do not specify SECLEN, the length of the secondary symptom string will not change from the length you specified on SYMRBLD INITIAL, or on a previous SYMRBLD RESET.

,VARLEN=*variable length*

Specifies the address of an optional halfword input variable that contains the maximum length in bytes of the variable data section. You can also directly specify a decimal digit for the length (for example, VARLEN=900). If you use register notation, the register contains the address of the length rather than the length itself.

The following formula calculates the length of the variable data section:

```
The length provided must be the total length of the variable data items
+ the number of items (x) 4.
```

(The 4 is for the 2 byte key + 2 bytes for the length.) Note that the maximum size of the entire symptom record is 1900 bytes.

If you do not specify VARLEN, the length of the variable data section will not change from the length you specified on SYMRBLD INITIAL, or on a previous SYMRBLD RESET.

Example

The following is an example of invoking SYMRBLD to build a symptom record:

- SYMRBLD INITIAL initializes sections 1 and 2 of the symptom record and provides component data for section 2.1.
- SYMRBLD PRIMARY stores the following primary symptom string data:
  - Program return code: PRCS/00028878
  - CSECT name: RIDS/ABE5698J
  - Load module name: RIDS/ABD5698J#L

  Note: The symptom PIDS/ABE5698J is automatically placed as the first symptom in the primary symptom string.
- SYMRBLD SECONDARY stores the following secondary symptom string data:
  - Module assembly level: LVLS/C20
  - Field name: FLDS/COUNTER
  - Value: VALU/HFFFFFFFF

- SYMRBLD VARIABLE stores additional data that can be used for debugging in section 5 of the symptom record.
- SYMRBLD COMPLETE indicates that the record is complete. INVOKE=YES indicates that the record is written to the logrec data set. by the SYMREC macro.

```
      SYMRBLD INITIAL,SR=SREC,
              PRIMLEN=100,SECLEN=50,VARLEN=50,
              ARCHLEV=10,COMPDSC=MYCOMP,
              PROGRAM=PROGNAME,PROGLEV=REL6,
              PROBLEM=MYPROB,
              SERVLEV=MYSERV

      SYMRBLD PRIMARY,SDBSTRING=S1_DATA

      SYMRBLD SECONDARY,SDBSTRING=S2_DATA,SDBKEY=SDBVALU_H,
              DATA=COUNTER,CONVERT=YES

      SYMRBLD VARIABLE,S5KEY=S5HEX,DATA=MYVARDAT

      SYMRBLD COMPLETE,INVOKE=YES

SREC     DS   CL600
MYCOMP   DC   CL13'COMPONENT XXX'
MYPROB   DC   CL14'DATABASE ERROR'
MYSERV   DC   CL9'VERSION 1'
PROGNAME DC   CL8'ABE5698J'
REL6     DC   CL3'REL6'
S1_DATA  DC   CL43'PRCS/00028878 RIDS/ABE5698J RIDS/ABD5698J#L'
S2_DATA  DC   CL22'LVLS/C20 FLDS/COUNTER'
MYVARDAT DC   XL2'01E4'
COUNTER  DC   X'FFFFFFFF'
```

SYMREC — Process a Symptom Record

Description

The SYMREC macro updates a symptom record with system environment information and then logs the symptom record in the logrec data set. The symptom record is a data area in the user's application that has been mapped by the ADSR mapping macro.

As an application detects errors during execution, it stores diagnostic information into the symptom record and issues the SYMREC macro to log the record. The diagnostic information consists of a description of a programming failure and a description of the environment in which the failure occurred.

When the SYMREC macro is invoked, it checks that all the required input fields of the ADSR symptom record are set by the caller. If the required input fields are not set, SYMREC issues appropriate return and reason codes.

The SYMREC macro can be used for authorized and unauthorized programs. Your installation controls which programs can write symptom records and whether to write the symptom record to the logrec data set, the job log, both or neither through an installation-written exit. This exit is called ASREXIT. For further information about ASREXIT, see *z/OS MVS Installation Exits*. SYMRBLD is a related macro. For more information see *z/OS MVS Programming: 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. If disabled, the input data to SYMREC must be in fixed storage or in disabled reference (DREF) storage.
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in the primary address space.

Programming Requirements

The caller must include the ADSR mapping macro to map the symptom record specified on the SR parameter. The caller must fill in this symptom record. For more information on the ADSR mapping macro, see *z/OS MVS Data Areas, Vol 1 (ABEP-DALT)*.

Restrictions

Although callers in 24-bit or 31-bit addressing mode can issue the SYMREC macro, the addresses passed to the SYMREC service must be 31-bit addresses.

Input Register Information

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

© Copyright IBM Corp. 1988, 2002

Register	Contents
13	The address of a standard 18-word save area

Output Register Information

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

Register	Contents
0	Reason code
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

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 SYMREC macro is written as follows:

---

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

SR=*addr*	*addr*: A-type address or register 2-12.

Parameters

The parameters are explained as follows:

SR=*addr*
Specifies the address of the symptom record. The SR parameter is required.

ABEND Codes

None.

Return and Reason Codes

When SYMREC returns control, registers 15 and 0 contain the following hexadecimal return codes and reason codes, respectively:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0000	0000	**Meaning**: SYMREC completed successfully and the symptom record was recorded. **Action**: None.
0004	0164	**Meaning**: Program error. An attempt to write section 1 information from the completed symptom record failed. The area was not accessible to a write request. The entire input record was recorded. **Action**: Make sure that the storage containing the input symptom record is not released before the SYMREC request completes.
0008	0158	**Meaning**: Program error. The total length of the input symptom record exceeds the maximum. A partial symptom record was recorded. **Action**: Correct the length of the symptom record. The maximum length of the symptom record is 1900 bytes. Sections 1, 2, and 2.1 of the symptom record are fixed in length. The length of sections 1, 2, and 2.1 combined is 212 bytes. Therefore, the combined length of sections 3, 4, and 5 must be less than or equal to 1688 bytes.
0008	015C	**Meaning**: Program error. Optional segments of the input symptom record were not accessible. The record includes the accessible entries of the input symptom record. A partial symptom record was recorded. **Action**: Verify that all optional sections (sections 4 and 5) of the symptom record are accessible.
000C	0104	**Meaning**: Program error. The first 2 bytes of the input symptom record do not contain the SR operand. No symptom record was recorded. **Action**: Verify that the correct address for the input symptom record was provided to the SYMREC service and that the first 2 bytes of the symptom record contain 'SR'.
000C	0108	**Meaning**: Program error. The input symptom record does not contain the required entries for section 2. No symptom record was recorded. **Action**: Make sure the following fields have been supplied in section 2 of the symptom record: the length of section 2 and the length/offset of section 2.1 and 3.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
000C	010C	**Meaning**: Program error. The input symptom record does not contain the required entries for section 2.1. No symptom record was recorded. **Action**: Make sure the following fields have been supplied in section 2.1 of the symptom record: section 2.1 identifier, architecture level of the symptom record, and the component release level or PID release level. Also verify that the length of section 2.1 is correct in section2.
000C	0114	**Meaning**: Program error. The input symptom record does not contain the required entries for section 3. No symptom record was recorded. **Action**: Make sure that the primary symptom string contains at least one symptom.
000C	0128	**Meaning**: Program error. This reason code is set when the input symptom record cannot be referenced. No symptom record was recorded. **Action**: Verify that the correct address for the symptom record was provided to the SYMREC macro and that this storage is accessible.
000C	012C	**Meaning**: Program error. All required sections of the symptom record could not be referenced. No symptom record was recorded. **Action**: Verify that all required sections (sections 1, 2, 2.1 and 3) of the symptom record are accessible.
000C	0134	**Meaning**: Program error. The input symptom record address is in non-accessible storage. No symptom record was recorded. **Action**: Verify the input parameter list provided to the SYMREC request.
000C	0144	**Meaning**: Program error. No symptom record was recorded. One of the following occurred: • The caller is in cross memory mode and the home address space is not accessible because it is swapped out or going through address space termination. **Action**: Make sure that the home address space is non-swappable during the SYMREC request. An address space can be made non-swappable using the SYSEVENT macro. • The caller is disabled, but it did not obtain MVS-recognized (valid) disablement. Valid disablement is obtained through a SETLOCK OBTAIN,TYPE=CPU request, available to supervisor state and key 0 callers only. **Action**: Use the SETLOCK OBTAIN, TYPE=CPU to disable normally.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
000C	0F1C	**Meaning**: Program error. The installation exit ASREXIT prevented the unauthorized caller from writing the symptom record to the logrec data set. No symptom record was recorded. **Action**: None. The installation has decided that unauthorized programs cannot write to the logrec data set.
0010	0F04	**Meaning**: Environmental error. There was insufficient space in the LOGREC buffer to accommodate the symptom record. No symptom record was recorded. **Action**: The request might be successful if retried. If the problem persists, record the return and reason code and supply it to the appropriate system support personnel.
0010	0F08	**Meaning**: System error. The SYMREC service could not acquire storage for a work area or a copy of the symptom record. No symptom record was recorded. **Action**: The request might be successful if retried. If the problem persists, record the return and reason code and supply it to the appropriate system support personnel.
0010	0F0C	**Meaning**: System error. Failure occurred while the symptom record was being moved to the LOGREC buffer. No symptom record was recorded. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.
0010	0F10	**Meaning**: System error. The SYMREC service has a logic error. No symptom record was recorded. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.
0010	0F14	**Meaning**: System error. The SYMREC service has shut itself down. It has exceeded the maximum allowable logic errors for the service routine. No symptom record was recorded. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.
0010	0F18	**Meaning**: System error. The SYMREC service has shut itself down. It has exceeded the maximum allowable incomplete SYMREC requests for processing. No symptom record was recorded. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.
0014	—	**Meaning**: System error. SYMREC is not operable. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.

SYMREC—List Form

Use the list form of the SYMREC 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 SYMREC macro is written as follows:

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

SR=*addr*	*addr*: A-type address (31 bit).
,MF=(L)	

Parameters

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

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

SYMREC—Execute Form

Use the execute form of the SYMREC 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 SYMREC macro is written as follows:

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

b	One or more blanks must follow SYMREC.

SR=*addr*	*addr*: A-type address (31 bit) or register 2-12. *addr*: A-type address (31 bit) or register 2-12. *addr*: A-type address (31 bit) or register 2-12.
,MF=(E,*list addr*)	*list addr*: RX-type address or register 2-12.

Parameters

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

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

SYNCH and SYNCHX — Take a Synchronous Exit to a Processing Program

Description

The SYNCH macro allows a program to take a synchronous exit to a processing program. After the processing program has finished, the program that issued the SYNCH macro regains control. The SYNCH macro is intended for use by primary mode programs only. If your program is in access register (AR) mode, use SYNCHX, which provides the same function as SYNCH.

Descriptions of the SYNCH and SYNCHX macro in this book are:

- The standard form of the SYNCH macro, which includes general information about the SYNCH and SYNCHX macros with specific information about the SYNCH macro. The syntax of the SYNCH macro and its parameters are explained.
- The standard form of the SYNCHX macro, which presents information specific to the SYNCHX macro. The topic explains the syntax of the SYNCHX macro and the parameters that are valid only on SYNCHX.
- The list form of the SYNCH and SYNCHX macros.
- The execute form of the SYNCH and SYNCHX macros.

Note

The SYNCH and SYNCHX macros have the same environment specifications, register information, programming requirements, restrictions and limitations, performance implications, and return and reason codes described below, except where noted in the explanation for SYNCHX.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit for SYNCH; 24- or 31- or 64-bit for SYNCHX.
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.

© Copyright IBM Corp. 1988, 2002

Input Register Information

Before issuing the SYNCH(X) 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	Values the processing program placed there before it returned to the caller
2-13	If RESTORE=YES, unchanged; if RESTORE=NO, values the processing program placed there before it returned to the caller
14	Used as a work register by the system
15	Value the processing program placed there before it returned to the caller

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 SYNCH macro is written as follows:

---

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

---

entry point addr	*entry point addr*: RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	**Default**: RESTORE=NO
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default**: AMODE=CALLER. **Note**: AMODE=DEFINED can be specified only if the entry point address is provided in a register.

Parameters

The parameters are explained as follows:

entry point addr
Specifies the address of the entry point of the processing program to receive control.

,RESTORE=NO
,RESTORE=YES
Specifies whether registers 2-13 are to be restored when control returns to the caller.

,AMODE=24
,AMODE=31
,AMODE=DEFINED
,AMODE=CALLER
Specifies the addressing mode in which the requested program is to receive control.

If AMODE=24 is specified, the requested program will receive control in 24-bit addressing mode.

If AMODE=31 is specified, the requested program will receive control in 31-bit addressing mode.

If AMODE=DEFINED is specified, the user must provide the entry point using a register and not an RX-type address. The requested program will receive control in the addressing mode indicated by the high order bit of the entry point address. If the bit is set to 0, the requested program will receive control in 24-bit addressing mode; if the bit is set to 1, the requested program will receive control in 31-bit addressing mode.

If AMODE=CALLER is specified, the requested program will receive control in the addressing mode of the caller.

Return and Reason Codes

None.

Example 1

Take a synchronous exit to PROGRAMA. Do not restore registers 2-13 when control returns.

```
LOAD   EP=PROGRAMA,DCB=LIB1     Load desired program
LR     R8,R0                    Obtain the entry point
SYNCH  (R8),RESTORE=NO
```

Example 2

Take a synchronous exit to a program labeled SUBRTN and restore registers 2-13 when control returns.

```
SYNCH  SUBRTN,RESTORE=YES
```

Example 3

Take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that this program is to execute in 24-bit addressing mode.

```
SYNCH (8),RESTORE=YES,AMODE=24
```

Example 4

Take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that this program is to receive control in the addressing mode defined by the high-order bit of its entry point address.

```
SYNCH (8),RESTORE=YES,AMODE=DEFINED
```

Example 5

Take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that this program is to receive control in the addressing mode as the caller.

```
SYNCH (8),RESTORE=YES,AMODE=CALLER
```

SYNCHX - Take a Synchronous Exit to a Processing Program

The SYNCHX macro provides the same function as the SYNCH macro. All parameters on the SYNCH macro are valid for the SYNCHX macro.

SYNCHX is intended for use by programs running in AR mode.

Note

The SYNCHX macro has the same environment specifications, register information, programming requirements, restrictions and limitations, performance implications, and return and reason codes as the SYNCH macro, except where noted below.

Environment

The SYNCHX macro can be used by callers in AR or primary ASC mode.

Programming Requirements

If your program is in AR mode, (1) issue the SYSSTATE ASCENV=AR macro before you issue SYNCHX, and (2) initialize AR 1 to zero.

Register Information

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

Register	Contents
0-13	Unchanged
14-15	Used as work registers by the system

Syntax

The SYNCHX macro is written as follows:

---

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

b — One or more blanks must precede SYNCHX.

SYNCHX

b	One or more blanks must follow SYNCHX.

---

entry point addr	*entry point addr*: RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	**Default**: RESTORE=NO
,AMODE=24 ,AMODE=31 ,AMODE=64 ,AMODE=DEFINED ,AMODE=CALLER	**Default**: AMODE=CALLER **Note:** AMODE=DEFINED can only be specified if the entry point is provided in a register. AMODE=DEFINED can only be used to SYNCHX to amode 24 and amode 31 programs.

Parameters

The parameters are described under the syntax of the standard form of the SYNCH macro.

SYNCH and SYNCHX—List Form

The list form of the SYNCH or SYNCHX macro is used to construct a control parameter list.

Syntax

The list form of the SYNCH or SYNCHX macro is written as follows:

---

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

,RESTORE=NO ,RESTORE=YES	**Default**: RESTORE=NO
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Default**: AMODE=CALLER

,MF=L

Parameters

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

,MF=L
Specifies the list form of the SYNCH or SYNCHX macro.

Example

Use the list form of the SYNCH macro to specify that registers 2-13 are to be restored when control returns from executing the SYNCH macro and that the addressing mode of the program is to be defined by the high-order bit of the entry point address. Assume that the execute form of the macro specifies the program address.

```
SYNCH ,RESTORE=YES,AMODE=DEFINED,MF=L
```

SYNCH and SYNCHX—Execute Form

The execute form of the SYNCH or SYNCHX macro uses a remote control-program parameter list that can be generated by the list form of SYNCH or SYNCHX.

Syntax

The execute form of the SYNCH or SYNCHX macro is written as follows:

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

entry point addr	*entry point addr*: RX-type address, or register (2) - (12) or (15).
,RESTORE=NO ,RESTORE=YES	
,AMODE=24 ,AMODE=31 ,AMODE=DEFINED ,AMODE=CALLER	**Note**: AMODE=DEFINED can be specified only if the entry point address is provided in a register.
,MF=(E,*ctrl addr*)	*ctrl addr*: RX-type address or register (1), (2) - (12).

Parameters

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

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

Example

Use the execute form of the SYNCH macro to take a synchronous exit to the program located at the address given in register 8 and restore registers 2-13 when control returns. Indicate that the program is to receive control in the same addressing mode as the caller and that the parameter list is located at SYNCHL2.

```
SYNCH (8),RESTORE=YES,AMODE=CALLER,MF=(E,SYNCHL2)
```

SYSEVENT — System Event

The SYSEVENT macro provides the interface to the system resource manager (SRM). Using SYSEVENT mnemonics, you can notify SRM of an event or ask SRM to perform a specific function. Out of the many different SYSEVENTs, only the following two are unauthorized:

QVS

REQFASD

See *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO* for more information on these unauthorized SYSEVENTs, as well as all of the authorized SYSEVENTs.

© Copyright IBM Corp. 1988, 2002

SYSSTATE — Identify System State

Description

Use the SYSSTATE macro 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 at the time the macro is issued.

For those macros that are sensitive to their environment, SYSSTATE identifies the environment. During the assembly stage, SYSSTATE sets one or more of the following:

- Global character symbol &SYSAM64, to identify the AMODE
- Global character symbol &SYSASCE, to identify the ASC mode
- Global arithmetic symbol &SYSALVL, to identify the Architectural level

Later, when the program is assembled, the macros check the global symbol(s) and generate the correct code. Table 3 on page 17 lists all the MVS macros and identifies the macros that need to know the environmental characteristics.

IBM recommends you issue SYSSTATE before you issue other macros. Once a program has issued SYSSTATE, there is no need to reissue it unless the program switches from one ASC mode to another or one AMODE 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

Another way to use the SYSSTATE macro is within a macro you write yourself. For example, you can issue SYSSTATE with the TEST parameter to ensure that the &SYSASCE global symbol has been set:

1. Define the &SYSASCE global symbol within your macro.
2. Issue SYSSTATE TEST, which sets &SYSASCE to the default if it has not yet been set.
3. Define different logical paths within your macro to correspond to the ASC mode that is in effect based on the value of &SYSASCE.

A program need use SYSSTATE TEST only when it wants to query the value of one of the variables. When setting variables (i.e., not SYSSTATE TEST), you can specify one or more of the parameters available. The variables associated with not-specified variables remain unchanged.

Environment

The requirements for the caller are:

Minimum authorization: Problem state, and any PSW key.

© Copyright IBM Corp. 1988, 2002

Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or 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

None.

Restrictions

None.

Input Register Information

Before issuing the SYSSTATE 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
0-15	Unchanged

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

Register	Contents
0-15	Unchanged
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The SYSSTATE macro is written as follows:

---

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

TEST

ASCENV=**P** ASCENV=AR	**Default**: ASCENV=P
AMODE64=**NO** AMODE64=YES	**Default:** AMODE64=NO
ARCHLVL=**0** ARCHLVL=1 ARCHLVL=2	**Default:** ARCHLVL=0

Parameters

The parameters are explained as follows:

TEST

TEST checks each one of the global symbols &SYSASCE, &SYSAM64, and &SYSALVL, and does the following for each as required:

- Sets the global symbol to its default, if the global symbol **does not** contain a value indicating that it had been set by a prior SYSSTATE macro.
- Leaves the global symbol unchanged, if the global symbol **does** contain a value indicating that you issued a specific SYSSTATE during this assembly.

ASCENV=P
ASCENV=AR

Indicates your program's ASC mode by setting the global symbol &SYSASCE.

- ASCENV=P indicates that the program is in primary mode.
- ASCENV=AR indicates that the program is in AR mode.

AMODE64=NO
AMODE64=YES

Indicates whether your program is AMODE 64 at this point. It sets the global symbol &SYSAM64.

- AMODE64=YES should be specified for any part of your program that runs in AMODE 64. Some macros process differently according to this specification. For example, macros such as ATTACHX, CALL, LINKX, LOAD, and XCTLX build prarmeter lists consisting of 8-byte entries when SYSSTATE AMODE64=YES.
- AMODE64=NO should be specified for programs, or parts of programs, that do not run in AMODE 64.

ARCHLVL=0
ARCHLVL=1
ARCHLVL=2

Indicates the architecture level of the system for which the subsequent section of your program is designed by setting the global symbol &SYSALVL.

- 0 means that the architecture is ESA/390. When bit CVTOS390_R10 in byte CVTOSLV2 of the CVT data area is off, your program should not assume that ARCHLVL=1 or ARCHLVL=2 capabilities are available.
- 1 means that the architecture is ESA/390 but includes the ESA/390 architecture items required by OS/390 Release 10 (e.g., the

relative/immediate instructions). When bit CVTOS390_R10 in byte CVTOSLV2 of the CVT data area is on, ARCHLVL=1 capabilities are available.

- 2 means that the architecture is z/Architecture. Macros that pay attention to ARCHLVL will avoid generating z/Architecture instructions when ARCHLVL < 2 is in effect. When byte FLCARCH in the PSA data area is not zero, ARCHLVL=1 and ARCHLVL=2 capabilities are available. If you set an ARCHLVL value less than the latest, your program can still run on more recent levels of the system, but it might not take advantage of all the new functions.

When ARCHLVL>0 is in effect, be aware that you still might need to use the IEABRC macro to have subsequent macros generate code that avoids the use of base-displacement branch instructions, since many macros are not sensitive to the value specified for SYSSTATE ARCHLVL. See *z/OS MVS Programming: Assembler Services Reference IAR-XCT* for details about the IEABRC macro.

ABEND Codes

None.

Return and Reason Codes

None.

Example 1

Change to AR mode and set the global symbol.

```
SAC      512
SYSSTATE ASCENV=AR
```

Example 2

Use SYSSTATE TEST within your own macro to check the global symbol and set it to the default if it is not already set.

```
         .
         .
         .
         GBLC     &SYSASCE                 Define global symbol
         SYSSTATE TEST                     If global symbol has no value,
                                           set to the default.
         AIF      ('&SYSASCE' EQ 'P').PR   Use code for primary ASC mode
 .AR     ANOP     This logical path contains instructions appropriate
                  for AR ASC mode.
         .
         .
         .
         AGO      .COMMON
 .PR     ANOP     This logical path contains instructions appropriate
                  for primary ASC mode.
         .
         .
         .
 .COMMON ANOP
```

TCBTOKEN — Request or Translate the TTOKEN

Description

The TTOKEN is the 16-byte identifier of a task. Unlike a TCB address, each TTOKEN is unique within the IPL; the system does not reassign this same identifier to any other TCB.

The TCBTOKEN macro provides three mutually exclusive services depending on how you specify the TYPE parameter:

- TYPE=CURRENT gives you the TTOKEN for the current task.
- TYPE=PARENT gives you the TTOKEN for the task that attached the current task.
- TYPE=JOBSTEP gives you the TTOKEN for the current task's job step task.

z/OS MVS Programming: Extended Addressability Guide describes TTOKENs.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	Any
AMODE:	31-bit
ASC mode:	Primary or 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:	Can reside in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the TCBTOKEN 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	Address of the TCBTOKEN parameter list
2-13	Unchanged
14	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	Used as a work register by the system
1	ALET used to address the TCBTOKEN parameter list
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 TCBTOKEN macro is written as follows:

---

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

TYPE=CURRENT TYPE=PARENT TYPE=JOBSTEP	
,TTOKEN=*ttoken addr*	*ttoken addr*: RX-type address.
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

TYPE=CURRENT
TYPE=PARENT
TYPE=JOBSTEP
Specifies the type of TCB information requested, as follows:

CURRENT
The system returns the TTOKEN of the currently active task. The TTOKEN is returned at the address specified by the TTOKEN parameter.

PARENT
The system returns the TTOKEN of the task that attached the currently active task. The TTOKEN is returned at the address specified by the TTOKEN parameter.

JOBSTEP
The system returns the TTOKEN of the job step task for the primary address space. The TTOKEN is returned at the address specified by the TTOKEN parameter.

,TTOKEN=*ttoken addr*
Specifies the address at which the 16-byte TTOKEN associated with the specified TCB is returned.

,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.

ABEND Codes

None.

Return Codes

When TCBTOKEN returns control, register 15 contains one of the following return codes:

Table 37. Return Codes for the TCBTOKEN Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: TCBTOKEN services completed successfully. **Action**: None.
10	**Meaning**: The TCB could not be referenced. **Action**: Ensure that the input TCB address is valid.
14	**Meaning**: The TCB did not pass the acronym check. **Action**: Ensure that the input TCB address is valid.
18	**Meaning**: The TCB has terminated. **Action**: None required.
20	**Meaning**: An unexpected error occurred. **Action**: Reissue TCBTOKEN.
24	**Meaning**: The contents of access register 1, used to address the parameter list, were not valid. **Action**: Change your program to run in primary mode or set access register 1 to zero.
28	**Meaning**: The parameter list is not valid. **Action**: Ensure that the parameter list address is valid and addressable in the calling program’s key.

Table 37. Return Codes for the TCBTOKEN Macro (continued)

Hexadecimal Return Code	Meaning and Action
30	**Meaning**: The task is scheduled for termination, but has not yet terminated. **Action**: None required.
34	**Meaning**: The caller is not running in task mode. **Action**: Change your program to run in task mode.

Example

Obtain the TTOKEN for the currently active task and store it in CURRENT_TTOKEN.

```
TCBTOKEN  TYPE=CURRENT,TTOKEN=CURRENT_TTOKEN
```

TCBTOKEN—List Form

The list form of the TCBTOKEN macro builds a nonexecutable parameter list that the execute form of the TCBTOKEN macro can refer to.

Syntax

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

---

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

,RELATED=*value*	*value*: Any valid macro parameter specification.
,MF=L	

Parameters

The parameters are explained below:

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

TCBTOKEN—Execute Form

The execute form of the TCBTOKEN macro modifies and executes the parameter list that the list form of TCBTOKEN generated.

Syntax

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

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

TYPE=CURRENT TYPE=PARENT TYPE=JOBSTEP	
,TTOKEN=*ttoken addr*	*ttoken addr*: RX-type address.
,RELATED=*value*	*value*: Any valid macro parameter specification.
,MF=(E,*cntl addr)*	*cntl addr*: RX-type address or register (1) - (12).

Parameters

The parameters are the same as those for the standard form of the TCBTOKEN macro with the following addition:

,MF=(E,*cntl addr*)
Specifies the execute form of the TCBTOKEN macro. This form uses a remote parameter list. The *cntl addr* specifies the address of the remote parameter list that the list form of the macro generates.

TESTART — Tests the Validity of ALETs

Description

TESTART tests for conditions that lead to an access register translation (ART) program interruption. Use it to test:

- The validity of an access list entry token (ALET)
- The validity of the extended authorization index (EAX) authority of the program that passed the ALET
- The value of an ALET
- If a specified ALET points to an entry for a SCOPE=COMMON data space.

By testing for these conditions, your program can avoid using an ALET that would cause an ART program interruption.

For information about ALETs, EAXs, and EAX-authorization, see *z/OS MVS Programming: Extended Addressability Guide*.

Environment

Requirements for the caller are:

Minimum authorization:	Problem state.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	Any
ASC mode:	Primary or AR
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks held:	No locks held
Control parameters:	Not applicable

Programming Requirements

None.

Restrictions

None.

Input Register Information

The input to the macro is the ALET and the caller's EAX.

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

© Copyright IBM Corp. 1988, 2002

2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

The TESTART macro is written as follows:

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

ALET=*(access-reg)*	*access-reg*: Access register (0) - (15).
,EAX=*(eax)*	*eax*: Register (0) - (14).
,CADS=YES ,CADS=NO	**Default**: CADS=NO

Parameters

The parameters are explained as follows:

ALET=*(access-reg)*
Specifies an access register 0 through 15 that contains the ALET to be tested.

,EAX=*(eax)*
Specifies a general purpose register 0 through 14 that contains the EAX to be used in the test, in bit positions 0-15. (The system ignores bits 16 - 31.)

,CADS=YES
,CADS=NO
Specifies if TESTART is to check the caller's PASN-AL to see if the specified ALET points to an entry for a SCOPE=COMMON data space. If CADS=YES is specified, TESTART returns *one* of the following return codes:
- X'04' if the ALET does not represent a SCOPE=COMMON data space
- X'18' if the ALET is for a SCOPE=COMMON data space.

If CADS=NO is specified, TESTART does not indicate whether or not the specified ALET is for a SCOPE=COMMON data space.

ABEND Codes

None.

Return Codes

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

Table 38. Return Codes for the TESTART Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The specified ALET is 0. **Action**: None.
04	**Meaning**: The specified ALET represents a valid entry on the DU-AL. If CADS=YES was specified on the call, the ALET does *not* point to an entry for a SCOPE=COMMON data space. **Action**: None required. However, you might take some action based upon your application.
08	**Meaning**: The specified ALET represents a valid entry on the PASN-AL. **Action**: None required. However, you might take some action based upon your application.
0C	**Meaning**: The specified ALET is 1. **Action**: None required. However, you might take some action based upon your application.
10	**Meaning**: The specified ALET and/or EAX will cause an ART program interruption. **Action**: None required. However, you might take some action based upon your application.
14	**Meaning**: A system error occurred in the TESTART service routine. **Action**: Retry the request.
18	**Meaning**: The program specified CADS=YES on the call to TESTART. The specified ALET points to an entry for a SCOPE=COMMON data space. **Action**: None required. However, you might take some action based upon your application.

Example 1

Request that TESTART verify the following two conditions:

- The ALET in AR1 passed by the caller is zero or is a valid ALET on the caller's dispatchable unit access list. The caller's registers were saved in the linkage stack prior to this example.
- The caller is EAX-authorized to data being passed as a parameter that can be accessed by the called program that runs with an authorized EAX.

```
R1       EQU   1                General register 1
AR1      EQU   1                Access register 1
R15      EQU   15               General register 15
*
         SLR   R15,R15          Set a zero code for the ESTA
         EREG  AR1,AR1          Extract GPR/AR 1 from the linkage stack
         ESTA  R0,R15           Place the caller's EAX in R1 bits 0-15
         TESTART ALET=(AR1),EAX=(R1)  Test the ALET/EAX
```

```
         CL    R15,=X'00000004' Test the TESTART return code
         BH    ERROR            Branch to error routine when the return
*                               code is greater than 4
```

Example 2

Request that TESTART verify the following two conditions:

- The ALET passed by the caller (on the linkage stack) points to an entry for a SCOPE=COMMON data space
- The caller is EAX-authorized to data being passed as a parameter that can be accessed by the called program that runs with an authorized EAX.

```
R1       EQU   1                General register 1
AR1      EQU   1                Access register 1
R15      EQU   15               General register 15
*
         SLR   R15,R15          Set a zero code for the ESTA
         EREG  AR1,AR1          Extract GPR/AR 1 from the linkage stack
         ESTA  R0,R15           Place the caller's EAX in R1 bits 0-15
         TESTART ALET=(AR1),EAX=(R1),CADS=YES  Test the ALET/EAX
         CL    R15,=X'00000018' Test the TESTART return code
         BE    CADS_ALET        Branch to CADS ALET routine processing
```

TIME — Obtain Time and Date

Description

The TIME macro returns either the local time of day and date, the Greenwich mean time of day and date, or the contents of the time-of-day (TOD) clock. The time-of-day clock referenced can be either in the basic time-of-day format (TOD) or the extended time-of-day format (ETOD).

- TOD — Unsigned 64-bit binary number
- ETOD — Unsigned 128-bit binary number

You can use the STCKCONV and CONVTOD macros to convert between TOD-clock format and various time of day and date formats. The STCKCONV macro converts a TOD-clock value to a time of day and date value and the CONVTOD macro converts a time of day and date value to a TOD clock value. See *z/OS MVS Programming: Assembler Services Guide* and *z/Architecture Principles of Operation* for information comparing the formats of the TOD and ETOD.

In a system using an external time reference (ETR[3]), the TOD clocks are set automatically at system initialization. However, in a system without an ETR, the time of day and date are only as accurate as the information entered by the operator. System response time also influences the accuracy of the values returned by the TIME macro.

There are two different linkage methods that can be specified. The TIME macro with LINKAGE=SYSTEM can be used by a program in primary or AR mode, in cross memory mode, and in either an enabled or disabled state. The LINKAGE=SYSTEM parameter also permits a choice of formats for the date value returned, as well as list and execute forms of the macro. With LINKAGE=SVC, the caller cannot be in cross memory mode or AR mode, must be in an enabled state, and has no choice of the format for the returned date value.

IBM recommends the use of the LINKAGE=SYSTEM parameter on the TIME macro. The LINKAGE=SVC parameter is provided solely for compatibility with existing programs.

The following description of the TIME macro is divided into two sections, LINKAGE=SYSTEM and LINKAGE=SVC. There are list and execute forms of the macro for LINKAGE=SYSTEM, but not for LINKAGE=SVC.

LINKAGE=SYSTEM

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 or PASN¬=HASN¬=SASN
AMODE:	24- or 31- or 64-bit addressing mode
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts

3. External time reference (ETR) is the MVS generic name for the IBM Sysplex Timer.

© Copyright IBM Corp. 1988, 2002

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 the program is in AR mode, issue the SYSSTATE ASCENV=AR macro before TIME. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

None.

Input Register Information

Before issuing the TIME 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	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 TIME macro with LINKAGE=SYSTEM is written as follows:

---

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

ƀ One or more blanks must precede TIME.

TIME

b	One or more blanks must follow TIME.

DEC,*stor addr* BIN,*stor addr* MIC,*stor addr* STCK,*stor addr* STCKE,*stor addr*	**Default**: DEC *stor addr*: RX-type address or register (0) or (2) - (12).
,ZONE=LT ,ZONE=GMT	**Default**: ZONE=LT **Note**: This parameter has no meaning if STCK or STCKE is specified.
,LINKAGE=SYSTEM	**Note**: LINKAGE=SVC is the default.
,DATETYPE=YYYYDDD ,DATETYPE=MMDDYYYY ,DATETYPE=DDMMYYYY ,DATETYPE=YYYYMMDD	**Default**: DATETYPE=YYYYDDD

Parameters

The parameters are explained as follows:

DEC,*stor addr*
BIN,*stor addr*
MIC,*stor addr*
STCK,*stor addr*
STCKE,*stor addr*

Specifies the format in which the time of day and date, or TOD clock contents, are returned. *stor addr* specifies the address of a 16-byte storage area in which TIME will return the values. The first two words of this area contain the time of day, or TOD clock contents, in the requested format. The third word contains the date in the requested format. Set the fourth word to zero before issuing TIME.

DEC returns the time of day as 8 bytes of packed decimal digits (without a sign) of the form

HHMMSSthmiju0000, where:

HH	is hours, based on a 24-hour clock
MM	is minutes
SS	is seconds
t	is tenths of seconds
h	is hundredths of seconds
m	is milliseconds
i	is ten-thousandths of seconds
j	is hundred-thousandths of seconds
u	is microseconds

BIN returns the time of day as an unsigned 32-bit binary number with the low-order bit equivalent to 0.01 second. The second word of the time value returned is zero.

MIC returns the time of day in microseconds. The value is returned as 8 bytes of information where bit 51 is equivalent to one microsecond.

STCK returns the contents of the basic TOD clock as an unsigned 64-bit binary number where bit 51 is equivalent to one microsecond.

STCKE returns the contents of the extended TOD clock (ETOD) as an unsigned 128-bit binary number where bit 59 is equivalent to one microsecond.

Note: The resolution of the time-of-day clock is model dependent. See *Principles of Operation* for an explanation of the rate advancement.

,ZONE=LT
,ZONE=GMT

LT specifies that the local time and date are to be returned. GMT specifies that an externally-sourced time and date such as Grenwich Mean Time (GMT) or Coordinated Universal Time (UTC) are to be returned. Refer to the section on time in *z/Architecture Principles of Operation*, SA22-7832 for a discussion of the differences between GMT and UTC.

ZONE is not meaningful if STCK or STCKE is specified.

,LINKAGE=SYSTEM

Specifies that non-SVC linkage is used to invoke the TIME service routine.

,DATETYPE=YYYYDDD
,DATETYPE=MMDDYYYY
,DATETYPE=DDMMYYYY
,DATETYPE=YYYYMMDD

Specifies the format in which the converted date is returned. For each parameter, the format of the returned date is as follows:

DATETYPE	Form of Returned Date
YYYYDDD	0YYYYDDD
MMDDYYYY	MMDDYYYY
DDMMYYYY	DDMMYYYY
YYYYMMDD	YYYYMMDD

The date is returned as packed decimal digits without a sign, where:

YYYY	is the year
DDD	is the day of the year
MM	is the month of the year
DD	is the day of the month

For example, with DATETYPE=YYYYDDD, January 21, 2000 would be returned as a converted TOD value of 02000021.

ABEND Codes

None.

Return Codes

When TIME macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code.

Table 39. Return Codes for the TIME Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Successful completion. **Action**: None.
04	**Meaning**: Programming error. TOD clocks are not initialized. **Action**: Retry the request later in the IPL.
08	**Meaning**: Environmental error. The TOD clock is not usable. **Action**: Retry the request.
0C	**Meaning**: System error. **Action**: Retry the request.
10	**Meaning**: Programming error. The user's parameter list is not in addressable storage. **Action**: Ensure that the parameter list is in the caller's Primary address space. If in AR mode, the PASN access list must not be used for addressing the parameter list.

Example 1

Request the local time of day and date (in year/day of the year format) to be returned in decimal digits in a 16-byte area called TIMEDATE.

```
          TIME  DEC,TIMEDATE,ZONE=LT,LINKAGE=SYSTEM
TIMEDATE  DS    CL16         TIME AND DATE RETURNED
```

Example 2

Request the GMT time of day and date to be returned in a 16-byte area called OUTVAL. The GMT time of day should be returned as microseconds and the date should be returned in a day/month/year format.

```
          TIME  MIC,OUTVAL,ZONE=GMT,LINKAGE=SYSTEM,DATETYPE=DDMMYYYY
OUTVAL    DS    CL16        TIME AND DATE RETURNED
```

LINKAGE=SYSTEM—List Form

Use the list form of the TIME macro (LINKAGE=SYSTEM) 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.

Syntax

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

---

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

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

LINKAGE=SYSTEM	**Note**: LINKAGE=SYSTEM must be specified in order to obtain the list form of the TIME macro.
,MF=L	

Parameters

The parameters are explained under the standard form of the TIME macro with LINKAGE=SYSTEM, with the following exception:

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

Example

Establish the correct amount of storage for the TIME parameter list.

```
LIST1     TIME  LINKAGE=SYSTEM,MF=L
```

LINKAGE=SYSTEM—Execute Form

Use the execute form of the TIME macro (LINKAGE=SYSTEM) 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 TIME macro is written as follows:

---

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

DEC,*stor addr* BIN,*stor addr* MIC,*stor addr* STCK,*stor addr*	**Default**: DEC *stor addr*: RX-type address or register (0) or (2) - (12).

STCKE,*stor addr*	
,ZONE=LT	**Default**: ZONE=LT
,ZONE=GMT	**Note**: This parameter has no meaning if STCK is specified.
,LINKAGE=SYSTEM	**Note**: LINKAGE=SYSTEM must be specified in order to obtain the execute form of the TIME macro.
,DATETYPE=YYYYDDD	**Default**: DATETYPE=YYYYDDD
,DATETYPE=MMDDYYYY	**Note**: This parameter has no meaning if STCK is specified.
,DATETYPE=DDMMYYYY	
,DATETYPE=YYYYMMDD	
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained under the standard form of the TIME macro with LINKAGE=SYSTEM, with the following exception:

,MF=(E,*list addr*)
Specifies the execute form of the TIME macro. *list addr* specifies the address of the parameter list created by the list form of the macro.

Example

Request the local time of day and date to be returned in a 16-byte area called OUTAREA. The local time of day should be returned as decimal digits and the local date should be returned in year/month/day format. Specify the address of the appropriate parameter list in LIST1.

```
         TIME  DEC,OUTAREA,LINKAGE=SYSTEM,MF=(E,LIST1),DATETYPE=YYYYMMDD
OUTAREA  DS    CL16          TIME AND DATE RETURNED
```

LINKAGE=SVC

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 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

None.

Restrictions

The caller cannot have any enabled, unlocked task (EUT) FRRs established.

Input Register Information

Before issuing the TIME 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 registers contain:

Register	Contents
0	The time of day if you specified DEC, BIN, or TU. If you did not specify any of these parameters, register 0 is used as a work register by the system.
1	Contains the date, if you specified DEC, BIN, TU, or MIC. If you did not specify any of these parameters, register 1 is used as a work register by the system.
2-13	Unchanged.
14	Used as a work register by the system.
15	Return code.

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

Performance Implications

None.

Syntax

The standard form of the TIME macro with LINKAGE=SVC is written as follows:

---

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

DEC BIN TU MIC,*stor addr* STCK,*stor addr*	**Default**: DEC *stor addr*: RX-type address or register (0) or (2) - (12).
,ZONE=LT ,ZONE=GMT	**Default**: ZONE=LT **Note**: This parameter has no meaning if STCK is specified.

,LINKAGE=SVC **Default**: LINKAGE=SVC

Note: The ERRET parameter is obsolete and will be ignored by the system. Therefore, the syntax and parameter descriptions for TIME no longer contain ERRET. However, the system will still accept ERRET and it is not necessary to delete it from existing code.

Parameters

The parameters are explained as follows:

DEC
BIN
TU
MIC,*stor addr*
STCK,*stor addr*

Specifies the form in which the time of day and date, or TOD clock contents, is returned.

DEC returns the time of day in register 0 as packed decimal digits, without a sign, 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

BIN returns the time of day in register 0 as an unsigned 32-bit binary number. The low-order bit is equivalent to 0.01 second.

TU returns the time of day in register 0 as an unsigned 32-bit binary number. The low-order bit is approximately 26.04166 microseconds (one timer unit).

MIC returns the time of day in microseconds. The *stor addr* is the address of an 8-byte area in storage with bit 51 equivalent to one microsecond.

STCK returns the contents of the TOD clock as an unsigned 64-bit binary number where bit 51 is equivalent to one microsecond. The *stor addr* is the address of an 8-byte area in storage.

Note: The resolution of the time-of-day clock is model dependent. See *Principles of Operation* for an explanation of the rate advancement.

The date is returned in register 1 as packed decimal digits of the form

0CYYDDDF, where:

C	is a digit representing the century. In the years 1900 through 1999, the macro will return a value of C=0. In the years 2000 through 2099, the macro will return a value of C=1.
YY	is the last two digits of the year.
DDD	is the day of the year.

F is a 4-bit sign character that allows the data to be unpacked and printed.

,ZONE=LT
,ZONE=GMT
Specifies that the local time and date (LT) or the Greenwich mean time and date (GMT) are to be returned.

,LINKAGE=SVC
Specifies that the linkage used to invoke the TIME service routine is through an SVC instruction.

ABEND Codes

10B

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

Return and Reason Codes

The only return code from the TIME macro is a zero in register 15 indicating successful completion.

Example 1

Request the system to store the time-of-day clock in the address pointed to by register 2.

```
TIME   STCK,(2)
```

Example 2

Request that the current local time and date be returned as packed decimal digits in registers 0 and 1.

```
TIME DEC,ZONE=LT,LINKAGE=SVC
```

Example 3

Request that the current time of day in microsecond format be returned in the location OUTAREA. Note that the default is taken for LINKAGE.

```
        TIME MIC,OUTAREA
        .
        .
OUTAREA DS   2F
```

TIMEUSED — Obtain Accumulated CPU or Vector Time

Description

The TIMEUSED macro returns an 8-byte hexadecimal number in a doubleword storage area that you specify. The number is the total CPU or vector time used by the current TCB up until you issue the macro. The format of the number is time-of-day (TOD) clock or microseconds time format.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	31- or 64-bit addressing mode
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space or be in an address space/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the TIMEUSED 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

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.

© Copyright IBM Corp. 1988, 2002

Syntax

The TIMEUSED macro is written as follows:

---

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

STORADR=*addr*	*addr*: RX-type address or register (2)-(12).
,LINKAGE=SYSTEM	
,RELATED=*value*	*value*: Any valid macro parameter specification
,CPU=TOD ,CPU=MIC ,VECTOR=TOD ,VECTOR=MIC	**Default**: CPU=TOD

Parameters

The parameters are explained as follows:

STORADR=*addr*
Specifies the 31-bit address of a doubleword area where the accumulated CPU or vector time is returned. The time interval is represented as an unsigned 64-bit binary number. If you specify CPU=TOD or VECTOR=TOD, bit 51 is the low-order bit of the interval value and equivalent to 1 microsecond. If you specify CPU=MIC or VECTOR=MIC, bit 63 is the low-order bit of the interval value and equivalent to 1 microsecond.

,LINKAGE=SYSTEM
Indicates that the linkage is by nonbranch entry.

,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.

,CPU=TOD
,CPU=MIC
,VECTOR=TOD

,VECTOR=MIC
Specifies that TIMEUSED should return the total CPU or vector time in either TOD clock format (CPU=TOD or VECTOR=TOD) or in microseconds (CPU=MIC or VECTOR=MIC).

ABEND Codes

The caller might encounter system completion code X'012'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

Return Codes

Register 15 contains one of the following hexadecimal return codes from TIMEUSED:

Table 40. Return and Reason Codes for the TIMEUSED Macro

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: The service completed successfully. **Action**: None.
8	**Meaning**: Unexpected error. **Action**: Reissue the TIMEUSED macro.

Example 1

Request the total CPU time in TOD clock format to be stored at the address in register 2.

```
TIMEUSED   STORADR=(2),LINKAGE=SYSTEM,CPU=TOD
```

Example 2

Request the total vector time in microseconds to be stored at the address in register 2.

```
TIMEUSED   STORADR=(2),LINKAGE=SYSTEM,VECTOR=MIC
```

TRANMSG — Translate Messages

Description

The TRANMSG macro returns a translated message or messages in a requested language. TRANMSG translates any of the following forms of messages:

- Self-defined text
- A message text block (MTB)
- A message parameter block (MPB)
- A combination of the above

TRANMSG uses a message input/output block (MIO) as input. You can either create the MIO, or let TRANMSG create it for you. You must create the MIO if you are translating multi-line messages with continuation lines. If you create the MIO for multi-line messages, it must contain the following:

- Code of the desired language
- Addresses of the messages to be translated
- Address of an output buffer in the calling program's address space into which TRANMSG is to return the translated messages.

You must also set the MIOCONT flag on in the MIO for multi-line messages with continuation lines.

Otherwise, use parameters on TRANMSG to provide that information, so TRANMSG can build the MIO correctly.

Upon return, each translated message is in the output buffer in the form of an MTB, and the MIO contains the addresses of the MTBs. If the translated message has more than one line, the MTB will indicate multiple lines by showing more than one message entry area within the MTB associated with the translated message.

See *z/OS MVS Programming: Assembler Services Guide* for more information on using TRANMSG.

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 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:	Not applicable

Programming Requirements

Before invoking TRANMSG, you must obtain storage for:

- The MIO
- The output buffer where TRANMSG will return the translated messages.

The size of the storage you will need for the MIO and output buffer depends on the number and size of messages you are translating. Refer to *z/OS MVS Data Areas,*

© Copyright IBM Corp. 1988, 2002

Vol 3 (IVT-RCWK) for a mapping of the MIO. Storage must be in the address space in which the calling program issued TRANMSG.

You must include the following mapping macros:
- CNLMMIO
- CNLMMCA

Restrictions

If TRANMSG builds the MIO for your application:
- Message translation starts at the first message in the message entry list (*list addr* in the INBUF parameter).
- The first message must contain a message identifier.
- You must supply all parameters on TRANMSG.

If you provide a formatted MIO, the only required parameter is MIO.

Input Register Information

Before issuing the TRANMSG macro, the caller must ensure that register 13 contains the address of an 18-word save area, which can be provided through the use of standard linkage conventions.

Output Register Information

When the TRANMSG macro returns control, the output registers contain the following values:

Register	Contents
0	The contents of the high-order halfword are not part of the intended programming interface. The low-order halfword contains a reason code.
1	Used as a work register by system
2-13	Unchanged
14	Used as a work register by system
15	Return code

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

Performance Implications

Translating multiple messages on one invocation of TRANMSG is more efficient than invoking TRANMSG multiple times with one message for each invocation.

Syntax

If you build the MIO, code the TRANMSG macro as follows:

---

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

b	One or more blanks must follow TRANMSG.

MIO=*msg block addr*	*msg block addr*: RX-type address or register (2) - (12).

If you want the TRANMSG macro to build the MIO, code TRANMSG as follows:

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

MIO=*msg block addr*	*msg block addr*: RX-type address or register (2) - (12).
,MIOL=*length of block addr*	*length of block addr*: RX-type address or register (2) - (12).
,INBUF=(*list addr, num of entries addr*)	*list addr*: RX-type address or register (2) - (12). *num of entries*: RX-type address or register (2) - (12).
,OUTBUF=*output buffer addr*	*output buffer addr*: RX-type address or register (2) - (12).
,OUTBUFL=*output buffer length addr*	*output buffer length addr*: RX-type address or register (2) - (12).
,LANGCODE=*lang code addr*	*lang code addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

MIO=*msg block addr*
Specifies the address, or a register, containing the address of an area containing the MIO or the address where TRANMSG is to build or find the MIO. If you have built the MIO, code only this parameter. Specify all other parameters only if TRANMSG is to build the MIO.

,MIOL=*length of block addr*
Specifies the address of a fullword or a register containing the length in bytes of the MIO. The length value is right-justified and padded with blanks. This parameter is required if TRANMSG is to build the MIO.

,INBUF=(*list addr, num of entries addr***)**
Specifies the address of a register pointing to the list of addresses of the self-defined text, MPB, or MTB that TRANMSG is to use as input, and the number of entries in the list, respectively. This parameter is required if TRANMSG is to build the MIO.

,OUTBUF=*output buffer addr*
Specifies the address of a register containing the address of the output buffer into which TRANMSG is to return translated messages in the form of MTBs. This parameter is required if TRANMSG is to build the MIO.

,OUTBUFL=*output buffer length addr*
Specifies the address of a fullword or a register containing the length in bytes of the output buffer. This parameter is required if TRANMSG is to build the MIO.

,LANGCODE=*lang code addr*
Specifies the address of, or a register pointing to, the 3-byte character field containing the code of the language into which you want the messages translated. *z/OS MVS Programming: Assembler Services Guide* contains a list of language codes. This parameter is required if TRANMSG is to build the MIO.

Return and Reason Codes

While TRANMSG provides return and reason codes in registers 15 and 0, respectively, you can determine exactly which message failed by looking at the reason code returned for each message in the MIOREAS field of the MIO variable data area. See *z/OS MVS Data Areas, Vol 3 (IVT-RCWK)* for a mapping of the MIO.

When TRANMSG completes, register 15 contains one of the following hexadecimal return codes:

Hexadecimal Return Code	Meaning
00	Processing completed successfully.
04	Processing complete. The output is complete, but TRANMSG might not have translated everything (for example, one variable in your message might not have translated).
08	Processing complete. The output is usable, but incomplete (for example, you might not have received all lines of a multiline message).
0C	Processing ended prematurely. The output is unusable. Possible causes are: • You have attempted to translate too many messages at one time. • The MIO is not valid • The output buffer is too small for any messages.
10	Processing did not complete. The output is unpredictable.

When TRANMSG completes, the low-order halfword of register 0 contains one of the following hexadecimal reason codes:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00	00	Successful processing.
04	07	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
04	08	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	0B	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	0C	The passed storage address is not valid.
04	0D	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	1A	TRANMSG returned a token value as text.
04	1B	The translated message is not a valid mixed DBCS string.
04	1C	A substitution token that is in the MPB is not in the message skeleton.
04	1D	A substitution token that is in the message skeleton is not in the MPB.
04	1F	The internal day code is not valid.
04	21	The required date format is not available. TRANMSG used the default.
04	22	A date formatting failure occurred.
04	23	The required time format is not available. TRANMSG used the default.
04	24	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	25	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
04	32	Input for the date format is not numeric. TRANMSG returned the date without formatting it.
08	01	The language you requested is not available. TRANMSG returned a U.S. English message.
08	03	The buffer space is insufficient for the output parameter blocks. The output was truncated.
08	14	The message identifier is longer than the text of the message continuation.
08	18	The input message length is not valid.
08	19	The input message does not match a message in the run-time message file.
08	1E	TRANMSG did not find a match in the target language run-time message file.
08	20	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.
08	2B	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
08	33	TRANMSG could not match the message ID in the message skeleton to those contained in the run-time message file.
08	34	TRANMSG attempted to match message text against an English message skeleton with translated line numbers. Input to TRANMSG must be an MPB when you use English message skeletons with translated line numbers.
0C	02	TRANMSG did not copy the input parameter block from the caller's address space.
0C	04	TRANMSG was unable to copy the MIO from the caller's address space.
0C	05	The MIO acronym is not valid.
0C	06	TRANMSG was unable to copy the MIO and output parameter blocks to the caller's address space.
0C	0A	TRANMSG could not obtain storage.
0C	10	The length of the MIO is less than the minimum length for a valid MIO.
0C	11	The length of the MTB is less than the minimum length for a valid MTB.
0C	12	The length of the MPB is less than the minimum length for a valid MPB.
0C	13	The MTB record count is not valid. The message record count must be one (1).
0C	15	The input message has a length less than three. A valid input message must have at least one character each for the message identifier and the message text, separated by a blank character.
0C	17	The MVS message service is unavailable.
0C	26	The translation request terminated. The MMS user exit has set the processing indicator to a nonzero value.
0C	27	The entry installation exit has failed.
0C	28	The exit installation exit has failed.
0C	29	The continuation ID in a multi-line message has zero length.
0C	2A	The MIO invocation type is not valid.
0C	31	The MIOXLATE field in the MIO is not valid.
0C	39	The MIO is too small.
0C	3A	The number in the list of entries is not a valid value.
10	09	This reason code is for internal diagnostic purposes only. Record it and supply it to the appropriate IBM support personnel.

If you translate multiple lines of message text

The return code and reason code you receive will reflect the most severe condition. Multiple lines of message text can be either multi-line messages or multiple messages. You will need to check the MIOREASN field contained within the

variable message entry areas of the MIO to determine processing status of each line. The MIOREASN field provides reasons for the errors.

If you received return codes 0 or 4, check field MIOTRUNC in the MIO to see if TRANMSG processed all message input.

It is possible that the output buffer was not large enough to hold all the translated messages. A return code of 0 or 4 might indicate this situation. Check the MIOTRUNC field of the MIO. If MIOTRUNC is 0, TRANMSG processed all messages. If MIOTRUNC is nonzero, it contains the number of the first message that did not fit into the input buffer.

If TRANMSG processing ended prematurely

You can increase the output buffer size, then reissue TRANMSG, or you can redrive message translation (that is, restart message translation at the point where it ended.) You can redrive message translation by using the same MIO and input and output data areas. Save the output of the failing message translation before redriving because TRANMSG reuses these fields on subsequent calls to translate the remaining messages. To redrive message translation, do the following:

1. First, determine where processing stopped. The nonzero number in the MIOTRUNC field is the number of the output message TRANMSG truncated because it did not fit into the output buffer. For example, if you issue TRANMSG to return five translated messages, and the output buffer can hold only three messages, TRANMSG will not return the fourth and fifth message in the output buffer. When TRANMSG completes, the MIOTRUNC field would contain a value of 4.
2. Set the MIOXLATE field of the MIO to the value of the MIOTRUNC field; in this case, 4.
3. If the first message to be translated is a continuation message (contains no message ID), also set the MIOMID field to the message value, and the MIOMIDL field to the message ID length of the associated continuation message.
4. Issue TRANMSG again to translate the remaining messages, starting, in this case, with the fourth message.

Repeat this process until MIOTRUNC is 0, indicating that all input messages have been processed.

If you don’t want to redrive using the same MIO, allocate a new, larger output buffer, change the MIO output buffer pointer, the length fields MIOBFPTR and MIOBFSIZ, and the MIOXLATE field. Issue TRANMSG again until MIOTRUNC is 0.

Example 1

Translate U.S. English text to Japanese using self-defined text as input. TRANMSG will build the MIO.

```
TRANSSDT CSECT
TRANSSDT AMODE  31
TRANSSDT RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
         ST     13,SAVE+4
         LA     15,SAVE
         ST     15,8(13)
         LR     13,15
```

```
*********************************************************************
*        GETMAIN STORAGE AREA FOR THE MIO                           *
*********************************************************************
*                                                                   *
         GETMAIN RU,LV=STORLEN,SP=SP230
         LR    R4,R1              SAVE STORAGE ADDRESS
         USING MIO,R4
         L     R2,MLENGTH         OBTAIN LENGTH OF MIO AREA
         AR    R2,R1              CALCULATE ADDRESS OF OUTPUT BUFFER
*                                                                   *
*********************************************************************
*        ISSUE TRANSLATE FOR MESSAGE                                *
*********************************************************************
*                                                                   *
         TRANMSG MIO=MIO,MIOL=MLENGTH,INBUF=(SDTA,ONE),              C
               OUTBUF=(R2),OUTBUFL=OUTAREAL,LANGCODE=LC
*********************************************************************
*        FREE STORAGE AREA FOR THE MIO                              *
*********************************************************************
*                                                                   *
         FREEMAIN RU,LV=STORLEN,SP=SP230,A=(4)
*                                                                   *
*********************************************************************
         L     13,SAVE+4
         LM    14,12,12(13)
         BR    14
         DROP
*********************************************************************
MLENGTH  DC    A(MLEN)
OUTAREAL DC    A(STORLEN-MLEN)
SDT      DC    H'37'
         DC    CL37'XXXX01 ENGLISH MESSAGE WITH ID XXXX01'
SDTA     DC    A(SDT)
LC       DC    CL3'JPN'
SP230    EQU   230
ONE      DC    F'1'
SAVE     DC    18F'0'
R1       EQU   1
R2       EQU   2
R4       EQU   4
MLEN     EQU   (MIOVDAT-MIO)+MIOMSGL
STORLEN  EQU   512
*********************************************************************
         DSECT
         CNLMMCA
         CNLMMIO
         END TRANSSDT
```

Example 2

Translate U.S. English text to Japanese. Build your own MIO.

```
TRANS2A  CSECT
TRANS2A  AMODE  31
TRANS2A  RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
         ST     13,SAVE+4
         LA     15,SAVE
         ST     15,8(13)
         LR     13,15
*                                                                   *
*********************************************************************
*        GETMAIN STORAGE AREA                                       *
*********************************************************************
*                                                                   *
         GETMAIN RU,LV=STORLEN,SP=SP230
```

```
         LR    R4,R1
         XC    0(MIOVDAT-MIO,R4),0(R4)         CLEAR MIO HEADER SECTION
         MVC   MIOACRN-MIO(L'MIOACRN,R4),=C'MIO ' SET ACRONYM
         MVI   MIOVRSN-MIO(R4),$MIO_VERSION       SET VERSION NUMBER
         MVC   MIOSIZE-MIO(4,R4),MLENGTH          SAVE MIO SIZE
         MVC   MIOLANG-MIO(L'MIOLANG,R4),=C'JPN'  SET LANGUAGE NAME
         L     R3,MLENGTH                         CALCULATE OUTAREA ADD
         AR    R3,R4                              GET MIO ADDRESS
         ST    R3,MIOBFPTR-MIO(,R4)               SET OUTAREA ADDRESS
         MVC   MIOBFSIZ-MIO(L'MIOBFSIZ,R4),OUTAREAL SET OUTAREA LENGTH
         LA    R3,1
         ST    R3,MIOXLATE-MIO(,R4)               SET TO FIRST MSG
         MVI   MIOMID-MIO(R4),C' '                INIT MSGID TO SPACES
         MVC   MIOMID-MIO+1(L'MIOMID-1,R4),MIOMID-MIO(R4)
         LA    R3,MIOMSGL                         GET LENGTH OF MIO
         ST    R3,MIOVDATL-MIO(,R4)         SAVE VARIABLE AREA LENGTH
         LA    R3,1
         ST    R3,MIOMSGNO-MIO(,R4)               SET NUMBER OF MSGS    C
                                                     TO TRANSLATE
         LA    R3,MIOVDAT-MIO                     GET OFFSET TO VAR. AREA
         ST    R3,MIOOFFST-MIO(,R4)               SAVE OFFSET TO 1ST MSG
         AR    R3,R4                         POINT TO MIO VARIABLE AREA
         XC    0(MIOMSGL,R3),0(R3)                CLEAR MSG ENTRY AREA
         LA    R2,SDT                        OBTAIN INPUT AREA ADDRESS
         ST    R2,MIOINPTP-MIOMSG(,R3)            SAVE INPUT AREA ADDRESS
         MVI   MIOINFL-MIOMSG(R3),MIOXLATF        INDICATE TRANSLATE
*                                                                      *
************************************************************************
*        ISSUE TRANSLATE FOR MESSAGE                                   *
************************************************************************
*                                                                      *
         TRANMSG MIO=(R4)
*                                                                      *
************************************************************************
*        FREE STORAGE AREA                                             *
************************************************************************
*                                                                      *
         FREEMAIN RU,LV=STORLEN,SP=SP230,A=(4)
*                                                                      *
************************************************************************
         L     13,SAVE+4
         LM    14,12,12(13)
         BR    14
         DROP
************************************************************************
         DS    0F
MLENGTH  DC    A(MLEN)
OUTAREAL DC    A(STORLEN-MLEN)
SDT      DC    H'37'
         DC    CL37'XXXX01 ENGLISH MESSAGE WITH ID XXXX01'
INAREA   DC    A(SDT)
LC       DC    CL3'JPN'
SP230    EQU   230
ONE      DC    F'1'
SAVE     DC    18F'0'
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
MLEN     EQU   (MIOVDAT-MIO)+MIOMSGL
STORLEN  EQU   512
************************************************************************
         DSECT
         CNLMMCA
         CNLMMIO
         END TRANS2A
```

Example 3

Translate three single-line U.S. English messages to Japanese using self-defined text as input.

```
TRANMULT CSECT
TRANMULT AMODE  31
TRANMULT RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
         ST     13,SAVE+4
         LA     15,SAVE
         ST     15,8(13)
         LR     13,15
*                                                                     *
***********************************************************************
*        GETMAIN STORAGE AREA                                         *
***********************************************************************
*                                                                     *
         GETMAIN RU,LV=STORLEN,SP=SP230
         LR    R4,R1                     SAVE STORAGE ADDRESS
         USING MIO,R4
         L     R2,MLENGTH                OBTAIN LENGTH OF MIO AREA
         AR    R2,R1                CALCULATE ADDRESS OF OUTPUT BUFFER
*                                                                     *
***********************************************************************
*        ISSUE TRANSLATE FOR MESSAGE                                  *
***********************************************************************
*                                                                     *
         TRANMSG MIO=MIO,MIOL=MLENGTH,INBUF=(SDT1A,THREE),             C
               OUTBUF=(R2),OUTBUFL=OUTAREAL,LANGCODE=LC
***********************************************************************
*        FREE STORAGE AREA                                            *
***********************************************************************
*                                                                     *
         FREEMAIN RU,LV=STORLEN,SP=SP230,A=(4)
*                                                                     *
***********************************************************************
         L     13,SAVE+4
         LM    14,12,12(13)
         BR    14
         DROP
***********************************************************************
MLENGTH  DC    A(MLEN)
OUTAREAL DC    A(STORLEN-MLEN)
SDT1     DC    H'33'
         DC    CL33'XXXX0A THIS IS MESSAGE NUMBER ONE'
SDT2     DC    H'33'
         DC    CL33'XXXX0B THIS IS MESSAGE NUMBER TWO'
SDT3     DC    H'35'
         DC    CL35'XXXX0C THIS IS MESSAGE NUMBER THREE'
SDT1A    DC    A(SDT1)
SDT2A    DC    A(SDT2)
SDT3A    DC    A(SDT3)
LC       DC    CL3'JPN'
SP230    EQU   230
THREE    DC    F'3'
SAVE     DC    18F'0'
R1       EQU   1
R2       EQU   2
R4       EQU   4
MLEN     EQU   (MIOVDAT-MIO)+(3*MIOMSGL)
STORLEN  EQU   512
***********************************************************************
```

```
         DSECT
         CNLMMCA
         CNLMMIO
         END TRANMULT
```

Example 4

Translate U.S. English text to Japanese using an MTB as input. Create the input MTB.

```
TRANMTBA CSECT
TRANMTBA AMODE  31
TRANMTBA RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
         ST     13,SAVE+4
         LA     15,SAVE
         ST     15,8(13)
         LR     13,15
*                                                                     *
***********************************************************************
*        GETMAIN STORAGE AREA                                         *
***********************************************************************
*                                                                     *
         GETMAIN RU,LV=STORLEN,SP=SP230
         LR    R4,R1              SAVE STORAGE ADDRESS
         USING MIO,R4
         L     R2,MLENGTH         OBTAIN LENGTH OF MIO AREA
         AR    R2,R4              CALCULATE ADDRESS OF MTB
         USING MTB,R2
         MVC   MTBACRN,=C'MTB '      SET ACRONYM
         MVI   MTBVRSN,$MTB_VERSION  SET VERSION NUMBER
         MVC   MTBLNGCD,LC           SET LANGUAGE CODE
         LA    R3,MTBLEN             CALCULATE SIZE OF MTB
         ST    R3,MTBSIZE            SAVE MTB SIZE
         LA    R3,MTBVDAT-MTB        OBTAIN LENGTH OF MTB HEADER
         ST    R3,MTBOFFST           SAVE OFFSET TO MTB VARIABLE AREA
         MVC   MTBCOUNT,ONE          SAVE RECORD COUNT
         MVC   MTBVDATL,SDTLEN       SAVE MTB VARIABLE AREA SIZE
         AR    R3,R2                 POINT TO MTB VARIABLE AREA
         USING MTBMSG,R3
         MVC   MTBMSG(39),SDT        SET MESSAGE LENGTH
         ST    R2,LIST               SAVE MTB ADDRESS LIST
         LA    R3,39(,R3)            SAVE ADDRESS OF OUTPUT BUFFER
***********************************************************************
*        ISSUE TRANSLATE FOR MESSAGE                                  *
***********************************************************************
*                                                                     *
         TRANMSG MIO=MIO,MIOL=MLENGTH,INBUF=(LIST,ONE),               C
               OUTBUF=(R3),OUTBUFL=OUTAREAL,LANGCODE=LC
***********************************************************************
*        FREE STORAGE AREA                                            *
***********************************************************************
*                                                                     *
         FREEMAIN RU,LV=STORLEN,SP=SP230,A=(4)
*                                                                     *
***********************************************************************
         L     13,SAVE+4
         LM    14,12,12(13)
         BR    14
***********************************************************************
MLENGTH  DC    A(MLEN)
OUTAREAL DC    A(STORLEN-(MLEN+MTBLEN))
SDT      DC    H'37'
         DC    CL37'XXXX01 ENGLISH MESSAGE WITH ID XXXX01'
LC       DC    CL3'JPN'
```

```
SP230    EQU   230
ONE      DC    F'1'
ZERO     DC    F'0'
SDTLEN   DC    F'39'
SAVE     DC    18F'0'
LIST     DC    F'0'
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
STORLEN  EQU   512
MLEN     EQU   (MIOVDAT-MIO)+MIOMSGL
MTBLEN   EQU   (MTBVDAT-MTB)+39
*******************************************************************
         DSECT
         CNLMMCA
         CNLMMIO
         CNLMMTB
         END TRANMTBA
```

Example 5

Translate a U.S. English multiline message into Japanese. Create the MIO.

```
TRANSMLA CSECT
TRANSMLA AMODE  31
TRANSMLA RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
         ST     13,SAVE+4
         LA     15,SAVE
         ST     15,8(13)
         LR     13,15
*                                                                 *
*******************************************************************
*        GETMAIN STORAGE AREA                                     *
*******************************************************************
*                                                                 *
         GETMAIN RU,LV=STORLEN,SP=SP230
         LR    R4,R1
         XC    0(MIOVDAT-MIO,R4),0(R4)       CLEAR MIO HEADER SECTION
         MVC   MIOACRN-MIO(L'MIOACRN,R4),=C'MIO ' SET ACRONYM
         MVI   MIOVRSN-MIO(R4),$MIO_VERSION       SET VERSION NUMBER
         MVC   MIOSIZE-MIO(4,R4),MLENGTH          SAVE MIO SIZE
         MVC   MIOLANG-MIO(L'MIOLANG,R4),=C'JPN'  SET LANGUAGE NAME
         L     R3,MLENGTH                         CALCULATE OUTAREA ADD
         AR    R3,R4                              GET MIO ADDRESS
         ST    R3,MIOBFPTR-MIO(,R4)               SET OUTAREA ADDRESS
         MVC   MIOBFSIZ-MIO(L'MIOBFSIZ,R4),OUTAREAL SET OUTAREA LENGTH
         LA    R3,1
         ST    R3,MIOXLATE-MIO(,R4)               SET TO FIRST MSG
         MVI   MIOMID-MIO(R4),C' '                INIT MSGID TO SPACE
         MVC   MIOMID-MIO+1(L'MIOMID,R4),MIOMID-MIO(R4) CLEAR MSGID
         LA    R3,MSGLEN                          GET LENGTH OF MIO
         ST    R3,MIOVDATL-MIO(,R4)          SAVE VARIABLE AREA LENGTH
         LA    R3,3
         ST    R3,MIOMSGNO-MIO(,R4)               SET NUMBER OF MSGS       C
                                                    TO TRANSLATE
         LA    R3,MIOVDAT-MIO                     GET OFFSET TO VAR. AREA
         ST    R3,MIOOFFST-MIO(,R4)               SAVE OFFSET TO 1ST MSG
         AR    R3,R4                         POINT TO MIO VARIABLE AREA
         LA    R15,MIOVDAT-MIO               GET LENGTH OF MIO HEADER
         AR    R15,R4                        GET ADDRESS OF MIO MSG ENTRY
         LA    R3,SDT1A                           GET MSG AREA LENGTH
         XC    0(MIOMSGL,R15),0(R15)              CLEAR MSG ENTRY AREA
         MVC   MIOINPTP-MIOMSG(4,R15),0(R3)       GET ADDRESS OF SDT
         MVI   MIOINFL-MIOMSG(R15),MIOXLATF       INDICATE TRANSLATE
```

```
         LA    R3,4(,R3)                    POINT TO NEXT MESSAGE ADDR.
         LA    R15,MIOMSGL(,R15)            POINT TO NEXT MESSAGE ENTRY
         L     0,TWO                             SET NUMBER OF MESSAGES
LOOP     DS    0H
         XC    0(MIOMSGL,R15),0(R15)             CLEAR MSG ENTRY AREA
         MVC   MIOINPTP-MIOMSG(4,R15),0(R3)      GET ADDRESS OF SDT
         OI    MIOINFL-MIOMSG(R15),MIOXLATF      INDICATE TRANSLATE
         OI    MIOINFL-MIOMSG(R15),MIOCONT       INDICATE CONTINUATION
         LA    R3,4(,R3)                    POINT TO NEXT MESSAGE ADDR.
         LA    R15,MIOMSGL(,R15)            POINT TO NEXT MESSAGE ENTRY
         BCT   0,LOOP                      LOOP UNTIL ALL MSGS PROCESSED
*                                                                      *
************************************************************************
*        ISSUE TRANSLATE FOR MESSAGE                                   *
************************************************************************
*                                                                      *
         TRANMSG MIO=(R4)
*                                                                      *
************************************************************************
*        FREE STORAGE AREA                                             *
************************************************************************
*                                                                      *
         FREEMAIN RU,LV=STORLEN,SP=SP230,A=(4)
*                                                                      *
************************************************************************
         L     13,SAVE+4
         LM    14,12,12(13)
         BR    14
************************************************************************
MLENGTH  DC    A(MLEN)
OUTAREAL DC    A(STORLEN-MLEN)
TWO      DC    F'2'
SDT1     DC    H'33'
         DC    CL33'MSGID1 ENGLISH MESSAGE - LINE ONE'
SDT2     DC    H'28'
         DC    CL28'ENGLISH MESSAGE - LINE TWO  '
SDT3     DC    H'30'
         DC    CL30'ENGLISH MESSAGE - LINE THREE  '
SDT1A    DC    A(SDT1)
SDT2A    DC    A(SDT2)
SDT3A    DC    A(SDT3)
LC       DC    CL3'JPN'
SAVE     DC    18F'0'
SP230    EQU   230
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R15      EQU   15
MSGLEN   EQU   3*MIOMSGL
MLEN     EQU   (MIOVDAT-MIO)+MSGLEN
STORLEN  EQU   512
************************************************************************
         DSECT
         CNLMMCA
         CNLMMIO
         END TRANSMLA
```

TTIMER — Test Interval Timer

Description

The TTIMER macro tests the timer interval established by an STIMER macro. It also optionally cancels the remaining time interval.

If MIC is specified, the remaining time is returned to the doubleword area specified in the address. Bit 51 of the area is the low-order bit of the interval value and equivalent to one microsecond. If a time interval has not been set or has already expired, the area is set to zero.

Note: The resolution of the timer is model dependent. See *Principles of Operation* for additional details concerning timing facilities.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31- or 64-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

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

Restrictions

Time intervals established via the STIMERM SET macro cannot be tested or cancelled with the TTIMER macro.

Input Register Information

Before issuing the TTIMER 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	Used as a work register by the system if you do not specify TU. If you specify TU, register 0 contains the amount of time remaining in a timer interval.
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

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 TTIMER macro is written as follows:

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

CANCEL	
,TU	**Default**: TU
,MIC,*stor addr*	*stor addr*: RX-type address, or register (0) or (2) - (12).

The ERRET parameter is obsolete and is ignored by the system. Therefore, the syntax and parameter descriptions for TTIMER no longer contain ERRET. However, the system still accepts ERRET and it is not necessary to delete it from existing code.

Parameters

The parameters are explained as follows:

CANCEL
Specifies that the remaining time interval and any exit routine are to be canceled. If the time interval has already expired, the CANCEL option has no effect and a value of zero time remaining is returned. In this case, a specified exit will still receive control. If a nonzero time remaining is returned when the CANCEL option is specified, any exit routine is canceled. If CANCEL is not designated, the unexpired portion of the time interval remains in effect.

If WAIT was coded in the STIMER macro that established the interval, the task is not taken out of the wait condition and CANCEL is ignored.

,TU
,MIC,*stor addr*
Specifies that the remaining time in the interval be returned.

For TU, the time is returned in register 0 as an unsigned 32-bit binary number. The low-order bit is approximately 26.04166 microseconds (one timer unit). If the time remaining is too great to be expressed in four bytes, the remaining time interval is set to the maximum possible value (X'FFFFFFFF') and the return code is set to 4.

For MIC, the time is returned in microseconds. The *stor addr* is the doubleword area on a doubleword boundary where the remaining interval is to be stored.

ABEND Codes

12E

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

Return Codes

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

Table 41. Return Codes for the TTIMER Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Successful completion. **Action**: None.
04	**Meaning**: You specified the TU parameter, but the time remaining is greater than X'FFFFFFFF'. **Action**: None required. However, you might take some action based upon your application.

Example 1

Cancel the task's current time interval. The time remaining, if any, should be returned in timer units in register 0.

```
TTIMER   CANCEL,TU
```

Example 2

Return the time remaining, in microseconds, to the storage location addressed by the label OUTAREA. Do not cancel the interval.

```
         TTIMER  ,MIC,OUTAREA
         .
         .
         DS    0D
OUTAREA  DC    2F
```

UCBDEVN — Return EBCDIC Device Number for a UCB

Description

Use the UCBDEVN macro to obtain the printable EBCDIC format for the device number of a given unit control block (UCB). When issuing UCBDEVN, an unauthorized caller must pass a copy of the UCB unless one of the following is true:

- The caller received the UCB address from an authorized program that can guarantee that the UCB is pinned or cannot be deleted by a dynamic configuration change.
- The caller is running in an environment where dynamic configuration changes cannot occur.
- The caller can otherwise guarantee that the UCB will not be deleted.

The caller can obtain a copy of the UCB by using the UCBSCAN macro. See *z/OS MVS Programming: Assembler Services Guide* for information about accessing UCBs.

Before issuing UCBDEVN, authorized callers must pin the UCB unless one of the following is true:

- The caller is running in an environment where dynamic configuration changes cannot occur
- The caller can otherwise guarantee that the UCB will not be deleted.

If you are coding an authorized program that must pin the UCB, see *z/OS MVS Programming: Authorized Assembler Services Guide* for information about accessing UCBs.

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:	No requirement

Programming Requirements

If you do not specify the UCBPTR parameter, you must include the IEFUCBOB mapping macro and establish addressability to the UCB common segment through a USING statement.

Restrictions

The caller of UCBDEVN cannot pass a copy of a UCB for a nonbase exposure of a multiple-exposure device. Multiple-exposure devices were supported prior to MVS/ESA SP 5.2.

© Copyright IBM Corp. 1988, 2002

When issuing UCBDEVN, the caller cannot pass a copy of an alias UCB of a parallel access volume.

UCBDEVN accepts above 16 megabyte UCBs, below 16 megabyte UCBs, and captured UCBs as input. To specify an above 16 megabyte UCB, the caller must run in AMODE 31. If the caller runs in AMODE 31 and passes a 24-bit UCB pointer, the pointer must have a clean high order byte.

Input Register Information

Before issuing the UCBDEVN macro, the caller must ensure that GPR 13 contains the address of an 18-word 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
14-15	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 UCBDEVN macro is written as follows:

---

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

DEVN=*devnumber*	*devnumber*: RS-type address.
,UCBPTR=*ucbptr*	*ucbptr*: RX-type address. **Note:** If you omit this parameter, the system assumes that you have established addressability to the UCB common segment.
,NONBASE=NO ,NONBASE=YES	**Default**: NO

Parameters

The parameters are explained as follows:

DEVN=*devnumber*
Specifies the name of the fullword area in which the system returns the EBCDIC device number.

,UCBPTR=*ucbptr*
Specifies a fullword containing the address of the UCB common segment, which contains the device number you need. If you omit this parameter, you must do the following:

- Include the IEFUCBOB mapping macro in your program to map the UCB.
- Establish addressability to the UCB common segment through a USING statement.
- Place the address of the UCB common segment in the register specified in the USING statement.

If the UCB common segment is for a multiple exposure device (supported on systems prior to MVS/ESA SP 5.2), the system returns printable EBCDIC for the base exposure device number.

,NONBASE=NO
,NONBASE=YES
Specifies which device number the caller should receive for a specified alias UCB of a parallel access volume. NO specifies the base device number, and YES specifies the alias device number.

Return and Reason Codes

UCBDEVN does not return any return codes.

Example

Use the UCBDEVN macro to obtain the printable EBCDIC form of the device number for the UCB whose address is in UCBVAL. The system is to return the value in the fullword named WORD1.

```
UCBDEVN DEVN=WORD1,UCBPTR=UCBVAL
```

UCBINFO — Return Information from a UCB

Description

Use the UCBINFO macro to obtain information from a unit control block (UCB) for a specified device. The UCBINFO macro provides the following options:

DEVCOUNT Returns a count of the UCBs for a device class or device group.

DEVINFO Returns information about a device, specifically, why the device is offline. For the base UCB of a &pav., DEVINFO returns the number of alias UCBs that are defined and the number that are usable.

PATHINFO Returns information about the device path and type of channel path associated with the device.

PATHMAP Returns information about the device path.

PRFXDATA Obtains a copy of the UCB prefix extension segment.

PAVINFO Returns information about the alias UCBs for a parallel access volume.

The options of the UCBINFO macro have the same environmental specifications, programming requirements, restrictions, register information, and performance implications described below, except where noted in the explanations of each option.

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 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: 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).

Programming Requirements

Before issuing the UCBINFO macro, you can issue the UCBSCAN macro to obtain the device number, which you must provide as input to UCBINFO. See *z/OS MVS Programming: Assembler Services Guide* for information about accessing UCBs.

The caller must include the appropriate mapping macro for the UCBINFO option being used:

Option	Mapping Macro
DEVCOUNT	None
DEVINFO	IOSDDEVI mapping macro
PATHINFO	IOSDPATH mapping macro
PATHMAP	IOSDMAP mapping macro
PRFXDATA	IOSDUPI mapping macro

© Copyright IBM Corp. 1988, 2002

PAVINFO IOSDPAVA mapping macro

See *z/OS MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.

Restrictions

None.

Input Register Information

Before issuing the UCBINFO 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; 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 work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

UCBINFO DEVCOUNT

Use the UCBINFO DEVCOUNT macro to obtain a count of the UCBs for a device class.

Syntax

The standard form of the DEVCOUNT option of the UCBINFO macro is written as follows:

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

DEVCOUNT

,COUNT=*count addr*	*count addr*: RS-type address or register (2) - (12).
,GROUP=DEVICELASS	
,DEVCLASS=ALL	**Default:** ALL
,DEVCLASS=CHAR	
,DEVCLASS=COMM	
,DEVCLASS=CTC	
,DEVCLASS=DASD	
,DEVCLASS=DISP	
,DEVCLASS=TAPE	
,DEVCLASS=UREC	
,GROUP=OTHER	
,DEVGROUP=PAVBASE	**Default:** PAVBASE
,DEVGROUP=PAVALIAS	
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
,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:

DEVCOUNT
Specifies that the system is to return a count of the UCBs.

,COUNT=*count addr*
Specifies the address of the fullword field that is to receive the count.

,GROUP=DEVICECLASS
GROUP specifies the grouping upon which the UCB count is based.

DEVICECLASS indicates that the UCB count is based on device classes.

,DEVICECLASS=ALL|CHAR|COMM|CTC|DASD|DISP|TAPE|UREC
Specifies the device class for which the corresponding UCBs are to be counted:

ALL	Counts UCBs for all device classes
CHAR	Counts UCBs for character reader device class
COMM	Counts UCBs for communications device class
CTC	Counts UCBs for channel to channel device class
DASD	Counts UCBs for direct access device class

DISP Counts UCBs for display device class
TAPE Counts UCBs for tape device class
UREC Counts UCBs for unit record device class

,GROUP=OTHER
GROUP specifies the grouping upon which the UCB count is based.

OTHER indicates that the UCB count is not based on device classes.

,DEVGROUP=PAVBASE
,DEVGROUP=PAVALIAS
Specifies the device group for which the corresponding UCBs are to be counted.

- **PAVBASE**, counts UCBs for Parallel Access Volume (PAV) base UCBs.
- **PAVALIAS**, counts UCBs for Parallel Access Volume (PAV) alias UCBs.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration 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**, 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.
- **2**, 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 2.

,RETCODE=*retcode addr*
Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*
Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO DEVCOUNT macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** The DEVCOUNT function completed successfully. **Action:** None.
08	01	**Meaning:** Program error. A caller in AR mode specified an ALET that was not valid. **Action:** Correct the ALET and reissue the macro.
08	02	**Meaning:** Program error. The system could not access the caller's parameter list. **Action:** Check to see if your program inadvertently overlaid the parameter list generated by the macro.
08	03	**Meaning:** Program error. The UCB address provided by the caller does not represent a valid UCB. **Action:** Correct the UCB address and reissue the macro.
08	05	**Meaning:** Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. **Action:** Correct the IOCTOKEN parameter.
0C	None	**Meaning:** Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. **Action:** Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.
20	None	**Meaning:** System error. An unexpected error occurred. **Action:** Supply the return code to the appropriate IBM support personnel.

Example

To invoke UCBINFO to return a count of all DASD devices, code:

```
         UCBINFO  DEVCOUNT,COUNT=CTAREA,DEVCLASS=DASD,           X
         RETCODE=INFORTCD,RSNCODE=RSNCD
         .
         .
         .

         DS  0D
CTAREA   DS  F
INFORTCD DS  F
RSNCD    DS  F
```

UCBINFO DEVCOUNT—List Form

Use the list form of the DEVCOUNT option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses 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.

The list form of the DEVCOUNT option of the UCBINFO macro is written as follows:

---

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

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
MF=(L,*list addr*)	*list addr*: RX-type address
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 UCBINFO DEVCOUNT with the following exceptions:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO DEVCOUNT 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.

UCBINFO DEVCOUNT—Execute Form

Use the execute form of the DEVCOUNT option of the UCBINFO 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 DEVCOUNT option of the UCBINFO macro is written as follows:

---

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

DEVCOUNT	
,COUNT=*count addr*	*count addr*: RS-type address or register (2) - (12).
,GROUP=DEVICELASS	
,DEVCLASS=ALL	**Default:** ALL
,DEVCLASS=CHAR	
,DEVCLASS=COMM	
,DEVCLASS=CTC	
,DEVCLASS=DASD	
,DEVCLASS=DISP	
,DEVCLASS=TAPE	
,DEVCLASS=UREC	
,GROUP=OTHER	
,DEVGROUP=PAVBASE	**Default:** PAVBASE
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
,RETCODE=*retcode addr*	*retcode addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO DEVCOUNT with the following exceptions:

,MF=(E,*list addr***)**
,MF=(E,*list addr***,COMPLETE)**
Specifies the execute form of the UCBINFO DEVCOUNT 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.

UCBINFO DEVINFO

Use the UCBINFO DEVINFO macro to obtain information about a device, specifically, reasons why the device is offline.

Syntax

The standard form of the DEVINFO option of the UCBINFO macro is written as follows:

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

DEVINFO	
,DEVIAREA=*deviarea addr*	*deviarea addr*: RX-type address or register (2) - (12).
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default:** IMPLIED_VERSION *plistver*: 2
,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:

DEVINFO
Specifies that the system is to return information about the specified UCB.

,DEVIAREA=*deviarea addr*
Specifies the address of a required 256-byte output field into which the system is to return information about the specified UCB. This field is mapped by the mapping macro IOSDDEVI.

,DEVN=*devn addr*
Specifies the address of a halfword that contains, in binary form, the device number of the device. The DEVN and UCBPTR parameters are mutually exclusive.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration 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**, 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.
- **2**, 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 2

,RETCODE=*retcode addr*
Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*
Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO DEVINFO macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** The DEVINFO function completed successfully. **Action:** None.
04	None	**Meaning:** Program error. No UCB exists for the device number specified in the DEVN parameter. **Action:** Correct the device number and reissue the macro.
08	01	**Meaning:** Program error. A caller in AR mode specified an ALET that was not valid. **Action:** Correct the ALET and reissue the macro.
08	02	**Meaning:** Program error. An error occurred when the system tried to access the caller's parameter list. **Action:** Ensure that you have met the environmental requirements for the macro, and reissue the macro.
08	03	**Meaning:** Program error. An unauthorized caller specified the UCBPTR parameter. The UCBPTR parameter can be specified by authorized callers only. **Action:** Specify the DEVN parameter instead of the UCBPTR parameter to indicate the device for which the system is to obtain information.
08	05	**Meaning:** Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. **Action:** Correct the IOCTOKEN parameter.
08	09	**Meaning:** Program error. An error occurred when the system attempted to reference the area specified by the DEVIAREA parameter. **Action:** Correct the address specified on the DEVIAREA parameter and reissue the macro.
0C	None	**Meaning:** Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. **Action:** Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
20	None	**Meaning:** System error. An unexpected error occurred. **Action:** Supply the return code to the appropriate IBM support personnel.
28	None	**Meaning:** Program error. The device number provided by the caller is an alias device number of a parallel access volume. For information about a parallel access volume, the caller must specify the base device number. **Action:** Correct the DEVN parameter and reissue the macro.

Example

To invoke UCBINFO to return device information, code:

```
         UCBINFO  DEVINFO,DEVIAREA=INFOAREA,DEVN=DEVNUM,         X
         RETCODE=INFORTCD
         .
         .
         .

         DS  0D
INFOAREA DS  CL256
INFORTCD DS  F
DEVNUM   DS  H
```

UCBINFO DEVINFO—List Form

Use the list form of the DEVINFO option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses 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.

The list form of the DEVINFO option of the UCBINFO macro is written as follows:

---

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

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2

MF=(L,*list addr*)	*list addr*: RX-type address
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 UCBINFO DEVINFO with the following exceptions:

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

UCBINFO DEVINFO—Execute Form

Use the execute form of the DEVINFO option of the UCBINFO 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 DEVINFO option of the UCBINFO macro is written as follows:

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

DEVINFO	
,DEVIAREA=*deviarea addr*	*deviarea addr*: RX-type address or register (2) - (12).
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
,RETCODE=*retcode addr*	*retcode addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO DEVINFO with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the UCBINFO DEVINFO 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.

UCBINFO PATHINFO

Use the UCBINFO PATHINFO macro to obtain information about the device path and type of channel path associated with the device.

Syntax

The standard form of the PATHINFO option of the UCBINFO macro is written as follows:

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

PATHINFO	
,PATHAREA=*patharea addr*	*patharea addr*: RX-type address or register (2) - (12).
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).

,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default:** IMPLIED_VERSION *plistver*: 2
,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:

PATHINFO
Specifies that the system is to return information about the device path and type of channel path for the specified UCB.

,PATHAREA=*patharea addr*
Specifies the address of the required 256-byte output field into which the system is to return information about the device path and type of channel path for the specified UCB. This field is mapped by the mapping macro IOSDPATH.

,DEVN=*devn addr*
Specifies the address of a halfword that contains, in binary form, the device number of the device.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration 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**, 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.

- **2**, 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 2

,RETCODE=*retcode addr*
Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*
Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PATHINFO macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** The PATHINFO function completed successfully. **Action:** None.
04	None	**Meaning:** Program error. No UCB exists for the device number specified in the DEVN parameter. **Action:** Correct the device number and reissue the macro.
08	01	**Meaning:** Program error. A caller in AR mode specified an ALET that was not valid. **Action:** Correct the ALET and reissue the macro.
08	02	**Meaning:** Program error. An error occurred when the system tried to access the caller's parameter list. **Action:** Ensure that you have met the environmental requirements for the macro, and reissue the macro.
08	03	**Meaning:** Program error. An unauthorized caller specified the UCBPTR parameter. The UCBPTR parameter can be specified by authorized callers only. **Action:** Specify the DEVN parameter instead of the UCBPTR parameter to indicate the device for which the system is to obtain path information.
08	05	**Meaning:** Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. **Action:** Correct the IOCTOKEN parameter.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	08	**Meaning:** Program error. An error occurred when the system attempted to reference the area specified by the PATHAREA parameter. **Action:** Correct the address specified on the PATHAREA parameter and reissue the macro.
0C	None	**Meaning:** Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. **Action:** Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.
18	04	**Meaning:** System error. The subchannel is in permanent error and cannot be accessed. **Action:** Supply the return and reason codes to the appropriate IBM support personnel.
18	08	**Meaning:** Environmental error. The UCB is not connected to a subchannel. **Action:** Verify that there is a device at the device number associated with the subchannel, and reissue the macro.
20	None	**Meaning:** System error. An unexpected error occurred. **Action:** Supply the return code to the appropriate IBM support personnel.

Example

To invoke UCBINFO to return device path and type of channel path information, code:

```
         UCBINFO  PATHINFO,PATHAREA=INFOAREA,DEVN=DEVNUM,         X
         RETCODE=INFORTCD
         .
         .
         .

         DS  0D
INFOAREA DS  CL256
INFORTCD DS  F
DEVNUM   DS  H
```

UCBINFO PATHINFO—List Form

Use the list form of the PATHINFO option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses 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.

The list form of the PATHINFO option of the UCBINFO macro is written as follows:

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

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
MF=(L,*list addr*)	*list addr*: RX-type address
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 UCBINFO PATHINFO with the following exceptions:

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

UCBINFO PATHINFO—Execute Form

Use the execute form of the PATHINFO option of the UCBINFO 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 PATHINFO option of the UCBINFO macro is written as follows:

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

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

PATHINFO	
,PATHAREA=*patharea addr*	*patharea addr*: RX-type address or register (2) - (12).
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
,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 address in register (2) - (12). **Default:** COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO PATHINFO with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the UCBINFO PATHINFO 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.

UCBINFO PATHMAP

Use the UCBINFO PATHMAP macro to obtain information about the device path.

Syntax

The standard form of the PATHMAP option of the UCBINFO macro is written as follows:

---

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

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

PATHMAP	
,MAPAREA=*maparea addr*	*maparea addr*: RX-type address or register (2) - (12).
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default:** IMPLIED_VERSION *plistver*: 2
,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:

PATHMAP
Specifies that the system is to return information about the device path for the specified UCB.

,MAPAREA=*maparea addr*
Specifies a required 40-byte field into which the system is to return information about the device path for the specified UCB. This field is mapped by the mapping macro IOSDMAP.

,DEVN=*devn addr*
Specifies the address of a halfword that contains, in binary form, the device number of the device.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration 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**, 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.
- **2**, 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 2

,RETCODE=*retcode addr*
Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*
Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PATHMAP macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** The PATHMAP function completed successfully. **Action:** None.
04	None	**Meaning:** Program error. No UCB exists for the device number specified in the DEVN parameter. **Action:** Correct the device number and reissue the macro.
08	01	**Meaning:** Program error. A caller in AR mode specified an ALET that was not valid. **Action:** Correct the ALET and reissue the macro.
08	02	**Meaning:** Program error. An error occurred when the system tried to access the caller's parameter list. **Action:** Ensure that you have met the environmental requirements for the macro, and reissue the macro.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	03	**Meaning:** Program error. An unauthorized caller specified the UCB common address in the MAPAREA field. Unauthorized callers cannot specify the UCB in MAPAREA. **Action:** Use the DEVN parameter instead of the MAPAREA field to indicate the device for which the system is to obtain path information.
08	05	**Meaning:** Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. **Action:** Correct the IOCTOKEN parameter.
08	06	**Meaning:** Program error. An error occurred when the system attempted to reference the area specified by the MAPAREA parameter. **Action:** Correct the address specified for MAPAREA and reissue the macro.
0C	None	**Meaning:** Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. **Action:** Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.
10	04	**Meaning:** System error. The subchannel is in permanent error and cannot be accessed. **Action:** Supply the return and reason code to the appropriate IBM support personnel.
20	None	**Meaning:** System error. An unexpected error occurred. **Action:** Supply the return code to the appropriate IBM support personnel.

Example

To invoke UCBINFO to return device path information, code:

```
         UCBINFO  PATHMAP,MAPAREA=INFOAREA,DEVN=DEVNUM,          X
         RETCODE=INFORTCD
         .
         .
         .

         DS  0D
INFOAREA DS  CL256
INFORTCD DS  F
DEVNUM   DS  H
```

UCBINFO PATHMAP—List Form

Use the list form of the PATHMAP option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses 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.

The list form of the PATHMAP option of the UCBINFO macro is written as follows:

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

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
MF=(L,*list addr*)	*list addr*: RX-type address
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 UCBINFO PATHMAP with the following exceptions:

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

UCBINFO PATHMAP—Execute Form

Use the execute form of the PATHMAP option of the UCBINFO 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 PATHMAP option of the UCBINFO macro is written as follows:

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

PATHMAP	
,MAPAREA=*maparea addr*	*maparea addr*: RX-type address or register (2) - (12).
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default:** IMPLIED_VERSION *plistver*: 2
,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 address in register (2) - (12). **Default:** COMPLETE

Parameters

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

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the UCBINFO PATHMAP 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.

UCBINFO PAVINFO

Use the UCBINFO PAVINFO macro to obtain selected information applicable to each exposure (base and alias) of a Parallel Access Volume (PAV).

Syntax

The standard form of the PAVINFO option of the UCBINFO macro is written as follows:

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

PAVINFO	
PAVINFOSUM=NO PAVINFOSUM=YES	**Default**: NO
,PAVAREA=*pavarea addr*	*pavarea addr*: RX-type address or register (2) - (12).
,PAVLEN=*pavarea length addr*	*pavarea lenth addr*: RX-type address or register (2) - (12).
,SCHINFO=NO ,SCHINFO=YES	**Default**: NO
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default:** IMPLIED_VERSION *plistver*: 2
,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:

PAVINFO

Obtain selected information that applies to each exposure of a Parallel Access Volume (PAV) device. The data returned by this function is an array. Depending on the input device, the following is returned:

- When the input device is a PAV-base, the first array entry represents the base and each subsequent array entry represents each of the bound PAV-alias devices associated with the base. Note that if the base has no bound PAV-aliases, then only the first array entry is filled in.
- When the input is a non-PAV DASD device, only the first array entry is filled in.

- When the input device is a PAV-alias or a non-DASD, a non-zero return code is returned.

PAVINFOSUM=NO
PAVINFOSUM=YES
Specifies whether to retrieve only a sum of channel measurement data and model dependent subchannel data for the base device and all of its aliases.

Note: The model dependent subchannel data is only retrieved if SCHINFO=YES.

NO Do not just retrieve a total of channel measurement data and model dependent subchannel data for the base device and all of its aliases. This option causes each element of the PAVA array to contain information for the base device and each of its aliases.

YES Retrieve only a sum of channel measurement data and model dependent subchannel data for the base device and all of its aliases. This option causes the first element of the PAVA array to contain information on the base device, however, the PAVACMB and PAVASMDB fields will contain totals for the base and all of its aliases.

,PAVAREA=*pavarea addr*
Specifies the address of a required output field into which the system will return information about the alias UCBs for the specified base device number . This field is mapped by the mapping macro IOSDPAVA.

,PAVLEN=*pavarea lengthaddr*
Specifies the address or a register containing the length of the area specified by the PAVAREA parameter.

,SCHINFO=NO
,SCHINFO=YES
Specifies whether to retrieve model-dependent subchannel data for the device.

NO Do not retrieve model-dependent subchannel data for the device.

YES Retrieve model-dependent subchannel data for the device.

,DEVN=*devn addr*
Specifies the address of a halfword that contains the base device number in binary form.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration 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**, 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.
- **2**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:
- IMPLIED_VERSION
- MAX
- A decimal value in the range of 1 - 3.

,RETCODE=*retcode addr*
Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*
Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PAVINFO macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** The PAVINFO function completed successfully. **Action:** None.
04	None	**Meaning:** Program error. No UCB exists for the device number specified in the DEVN parameter. **Action:** Correct the device number and reissue the macro.
08	01	**Meaning:** Program error. A caller in AR mode specified an ALET that was not valid. **Action:** Correct the ALET and reissue the macro.
08	02	**Meaning:** Program error. An error occurred when the system tried to access the caller's parameter list. **Action:** Ensure that you have met the environmental requirements for the macro, and reissue the macro.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	03	**Meaning:** Program error. An unauthorized caller specified the UCBPTR parameter. The UCBPTR parameter can be specified by authorized callers only. **Action:**Specify the DEVN parameter instead of the UCBPTR parameter to indicate the device for which the system is to obtain information.
08	05	**Meaning:** Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. **Action:** Correct the IOCTOKEN parameter and reissue the macro.
08	0A	**Meaning:** Program error. An error occurred when the system attempted to reference the area specified by the PAVAREA parameter. **Action:** Correct the address specified on the PAVAREA parameter and reissue the macro.
0C	None	**Meaning:** Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. **Action:** Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.
1C	01	**Meaning:** Program error. The device number provided by the caller specifies a device that is not a DASD or is a PAV alias device. **Action:** Correct the DEVN parameter and reissue the macro.
1C	02	**Meaning:** Program error. The work area specified with the PAVAREA parameter is not large enough to contain the minimum amount of data. No data is returned. **Action:** Increase the size of the specified work area and reissue the macro.
1C	03	**Meaning:** Program error. The work area specified with the PAVAREA parameter is not large enough to contain an array element for each alias device. **Action:** Increase the size of the specified work area and reissue the macro.
20	None	**Meaning:** System error. An unexpected error occurred. **Action:** Supply the return code to the appropriate IBM support personnel.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
28	None	**Meaning:** Program error. The device number provided by the caller is an alias device number of a parallel access volume. The caller must specify the base device number. **Action:** Correct the DEVN parameter and reissue the macro.

Example

To invoke UCBINFO to return information about alias UCBs for a base device number, code:

```
         UCBINFO PAVINFO,DEVN=DEVNUM,PAVAREA=INFOAREA,PAVLEN=AREALEN,  X
                RETCODE=INFORTCD
         .
         .
         .
         DS  0D
DEVNUM   DS  H
INFOAREA DS  CL256
AREALEN  DS  F
INFORTCD DS  F
```

UCBINFO PAVINFO—List Form

Use the list form of the PAVINFO option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses to contain the parameters.

The list form of the PAVINFO option of the UCBINFO macro is written as follows:

---

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

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
MF=(L,*list addr*)	*list addr*: RX-type address
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 UCBINFO PAVINFO with the following exceptions:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO PAVINFO 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.

UCBINFO PAVINFO—Execute Form

Use the execute form of the PAVINFO option of the UCBINFO 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 PAVINFO option of the UCBINFO macro is written as follows:

---

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

PAVINFO	
PAVINFOSUM=NO PAVINFOSUM=YES	**Default**: NO
,PAVAREA=*pavarea addr*	*pavarea addr*: RX-type address or register (2) - (12).
,PAVLEN=*pavarea length addr*	*pavarea lenth addr*: RX-type address or register (2) - (12).
,SCHINFO=NO ,SCHINFO=YES	**Default**: NO
,DEVN=*devn addr*	*devn addr*: RX-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	

,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 2
,RETCODE=*retcode addr*	*retcode addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO PAVINFO with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
Specifies the execute form of the UCBINFO PAVINFO 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.

UCBINFO PRFXDATA

Use the UCBINFO PRFXDATA macro to obtain a copy of the UCB prefix extension segment.

Syntax

The standard form of the PRFXDATA option of the UCBINFO macro is written as follows:

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

PRFXDATA	
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).
,UCBPAREA=*ucbparea addr*	*ucbparea addr*: RX-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).

,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	**Default:** IMPLIED_VERSION *plistver*: 2
,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:

PRFXDATA
Specifies that the system is to obtain information from the UCB prefix extension segment.

,DEVN=*devn addr*
Specifies the address of a halfword that contains, in binary form, the device number of the device.

,UCBPAREA=*ucbparea addr*
Specifies the address of a 48-character storage area into which the system copies the UCB prefix extension segment. The IOSDUPI mapping macro maps the area.

,IOCTOKEN=*ioctoken addr*
Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro. If the I/O configuration token that is current when UCBINFO is invoked does not match the token whose address is supplied here, the system issues a return code to the caller.

If you set the input IOCTOKEN (specified by *ioctoken addr*) to binary zeros, UCBINFO sets IOCTOKEN to the current I/O configuration 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**, 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.
- **2**, 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 2

,RETCODE=*retcode addr*
Specifies the address of a fullword field into which the system copies the return code from GPR 15.

,RSNCODE=*rsncode addr*
Specifies the address of a fullword field into which the system copies the reason code from GPR 0.

Return and Reason Codes

When the UCBINFO PRFXDATA macro returns control to your program, GPR 15 (or *retcode addr*, if you coded RETCODE) contains a return code, and GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** The PRFXDATA function completed successfully. **Action:** None.
04	None	**Meaning:** Program error. No UCB exists for the device number specified in the DEVN parameter. **Action:** Correct the device number and reissue the macro.
08	01	**Meaning:** Program error. A caller in AR mode specified an ALET that was not valid. **Action:** Correct the ALET and reissue the macro.
08	02	**Meaning:** Program error. An error occurred when the system tried to access the caller's parameter list. **Action:** Ensure that you have met the environmental requirements for the macro, and reissue the macro.
08	03	**Meaning:** Program error. An unauthorized caller specified the UCBPTR parameter. The UCBPTR parameter can be specified by authorized callers only. **Action:** Specify the DEVN parameter instead of the UCBPTR parameter to indicate the device for which the system is to obtain information.
08	05	**Meaning:** Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. **Action:** Correct the IOCTOKEN parameter.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0C	None	**Meaning:** Environmental error. The I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. **Action:** Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero.
20	None	**Meaning:** System error. An unexpected error occurred. **Action:** Supply the return code to the appropriate IBM support personnel.

Example

To invoke UCBINFO to obtain a copy of the UCB prefix extension segment, code:

```
         UCBINFO  PRFXDATA,DEVN=DEVNUM,UCBPAREA=UAREA,            X
         RETCODE=INFORTCD
         .
         .
         .

         DS  0D
DEVNUM   DS  H
UAREA    DS  CL48
INFORTCD DS  F
```

UCBINFO PRFXDATA—List Form

Use the list form of the PRFXDATA option of the UCBINFO macro together with the execute form for applications that require reentrant code. The list form of the macro defines an area of storage that the execute form uses 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.

The list form of the PRFXDATA option of the UCBINFO macro is written as follows:

---

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

ƀ — One or more blanks must precede UCBINFO.

UCBINFO

ƀ — One or more blanks must follow UCBINFO.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX — **Default:** IMPLIED_VERSION

,PLISTVER=*plistver*	*plistver*: 2
MF=(L,*list addr*)	*list addr*: RX-type address
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 UCBINFO PRFXDATA with the following exceptions:

MF=(L,*list addr*)
MF=(L,*list addr,attr*)
MF=(L,*list addr*,0D)

Specifies the list form of the UCBINFO PRFXDATA 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.

UCBINFO PRFXDATA—Execute Form

Use the execute form of the PRFXDATA option of the UCBINFO 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 PRFXDATA option of the UCBINFO macro is written as follows:

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

PRFXDATA	
,DEVN=*devn addr*	*devn addr*: RS-type address or register (2) - (12).

,UCBPTR=*ucbptr addr*	*ucbptr addr*: RS-type address or register (2) - (12). **Note:** Specify either DEVN or UCBPTR, but not both.
,UCBPAREA=*ucbparea addr*	*ucbparea addr*: RX-type address or register (2) - (12).
,IOCTOKEN=*ioctoken addr*	*ioctoken addr*: RX-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default:** IMPLIED_VERSION *plistver*: 2
,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 address in register (2) - (12). **Default:** COMPLETE

Parameters

The parameters are explained under the standard form of UCBINFO PRFXDATA with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBINFO PRFXDATA 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.

UCBSCAN — Scan UCBs

Description

Use the UCBSCAN macro to scan unit control blocks (UCBs) and return a copy of a UCB.

Two types of scans are available with UCBSCAN: A scan of all UCBs, and a scan of all UCBs within a particular device class. For each type of scan, the caller may optionally:

- Restrict the scan to UCBs defined as static or installation-static.
- Restrict the scan to UCBs with 3-digit device numbers.
- Request nonbase exposures of a multiple-exposure device, supported on systems prior to MVS/ESA SP 5.2.
- Request alias UCBs for a parallel access volume.
- Specify the device number with which the scan should begin.

UCBSCAN presents the UCBs in ascending device number order. On each invocation, UCBSCAN returns a copy of requested UCB segments and data in caller-supplied areas. See *z/OS MVS Programming: Assembler Services Guide* for information on accessing UCBs.

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- 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, 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).

Programming Requirements

If in AR mode, issue SYSSTATE ASCENV=AR before issuing UCBSCAN.

Restrictions

None.

Input Register Information

Before issuing the UCBSCAN 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:

© Copyright IBM Corp. 1988, 2002

Register	Contents
0	Reason code if GPR 15 contains a return code of 04 or 08; otherwise, used as a work register by the system
1	Used as a work register by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code

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

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

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

---

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

COPY

,WORKAREA=*workarea addr*	*workarea addr:* RX-type address or register (2) - (12).
,UCBAREA=*ucbarea addr*	*ucbarea addr:* RX-type address or register (2) - (12).
,CMXTAREA= *cmxtarea addr*	*cmxtarea addr:* RX-type address or register (2) - (12).
,CMXTAREA=NONE	**Default:** NONE
,UCBPAREA= *ucbparea addr*	*ucbparea addr:* RX-type address or register (2) - (12).
,UCBPAREA=NONE	**Default:** NONE
,DCEAREA= *dcearea addr*	*dcearea addr:* RX-type address or register (2) - (12).
,DCEAREA=NONE	**Default:** NONE
,DCELEN=*length addr*	*length addr:* RS-type address or register (2) - (12). **Note:** DCELEN is valid only with DCEAREA and is required with DCEAREA.
,VOLSER=*volser addr*	*volser addr:* RS-type address or register (2) - (12).
,VOLSER=NONE	**Default:** NONE

,DEVNCHAR= *devnchar addr*	*devnchar addr:* RS-type address or register (2) - (12).
,DEVN=*devn addr* ,DEVN=0	*devn addr:* RS-type address or register (2) - (12). **Default:** 0
,DYNAMIC=NO ,DYNAMIC=YES	**Default:** NO
,RANGE=3DIGIT ,RANGE=ALL	**Default:** 3DIGIT
,NONBASE=NO ,NONBASE=YES	**Default:** NO
,UNBOUND_ALIAS=NO ,UNOUND_ALIAS=YES	**Default:** NO
,DEVCLASS=ALL ,DEVCLASS=CHAR ,DEVCLASS=COMM ,DEVCLASS=CTC ,DEVCLASS=DASD ,DEVCLASS=DISP ,DEVCLASS=TAPE ,DEVCLASS=UREC	**Default:** ALL
,DEVCID=*devcid addr* ,DEVCID=0	*devcid addr:* RS-type address **Default:** 0
,IOCTOKEN=*ioctoken addr* ,IOCTOKEN=NONE	*ioctoken addr:* RX-type address or register (2) - (12). **Default:** NONE
,PLISTVER=IMPLIED_VERSION ,PLISTVER=MAX ,PLISTVER=*plistver*	 **Default:** IMPLIED_VERSION *plistver:* 1
,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:

COPY
Specifies that a copy of the UCB is to be obtained. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy.

Note: When you issue UCBSCAN to obtain a UCB copy, the UCBID field in the copy is set to x‘CC’.

,WORKAREA=*workarea addr*
Specifies the address of a 100-character work area used by the UCBSCAN

service. The caller must initialize this work area to binary zeros before starting a UCB scan. On subsequent invocations of UCBSCAN within the same scan, the caller must leave the contents of this work area unchanged.

,UCBAREA=*ucbarea addr*
Specifies the address of a 48-character storage area that will receive a copy of the UCB common segment and the UCB device-dependent segment. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy.

The caller does not need to initialize this area. Use the IEFUCBOB mapping macro to map the area. The contents of certain fields in the copy are:

- The UCBEXTP field contains either:
  - The address of the CMXTAREA, if CMXTAREA is below 16 MB
  - 0, if CMXTAREA is above 16 MB or if the CMXTAREA parameter is not specified
- The UCBNXUCB field is 0, because this field is not valid in the UCB copy.
- Address fields in the copy might not contain valid addresses, so do not use these addresses to reference the data areas they point to.

,CMXTAREA=*cmxtarea addr*
,CMXTAREA=NONE
Specifies the address of a 32-character storage area that will receive a copy of the UCB common extension segment. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy and require this segment as part of a UCB copy.

Use the UCBCMEXT DSECT in the IEFUCBOB mapping macro to map the area. If the CMXTAREA area is below 16 MB, the UCBEXTP field in the UCBAREA area contains the address of the CMXTAREA area, If the CMXTAREA area is above 16 MB, the caller must explicitly supply the address of the CMXTAREA area because the UCBEXTP field will contain 0.

The UCBIEXT field contains 0 because this field is not valid in the UCB copy.

The UCBCLEXT field contains the address of the DCEAREA if the UCB has a device class extension and the caller specified the DCEAREA parameter. Otherwise, the field contains 0.

,UCBPAREA=*ucbparea addr*
,UCBPAREA=NONE
Specifies the address of a 48-character storage area that will receive a copy of the UCB prefix extension segment. The area can be mapped by the IOSDUPI mapping macro.

,DCEAREA=*dcearea addr*
,DCEAREA=NONE
Specifies the address of a storage area that will receive a copy of the UCB device class extension segment. See *z/OS HCD Planning* for a list of the MVS services that accept a UCB copy and require this segment as part of a UCB copy.

Note: If DCEAREA=NONE is coded, then DCELEN=0 must be coded. If DCEAREA=NONE is defaulted, then DCELEN does not have to be coded.

,DCELEN=*length addr*
Specifies the address of a 2-byte field that contains the length of the area specified by DCEAREA. The length specified must be 1 through 256 bytes. DCELEN is required with DCEAREA.

,VOLSER=*volser addr*
,VOLSER=NONE
Specifies the address of a 6-character field that indicates, in EBCDIC, the volume serial number of the device for which a UCB copy is to be obtained.

,DEVNCHAR=*devnchar addr*
Specifies the address of a 4-character field that is to receive the EBCDIC device number associated with the UCB copy.

,DEVN=*devn addr*
,DEVN=0
Specifies (DEVN=*devn addr*) an input halfword that contains, in binary form, the device number with which the scan is to begin. The default, DEVN=0, starts the scan with the first UCB.

,DYNAMIC=NO
,DYNAMIC=YES
Specifies whether the scan should be restricted to static and installation-static UCBs (DYNAMIC=NO) or should also include dynamic UCBs (DYNAMIC=YES).

,RANGE=3DIGIT
,RANGE=ALL
Specifies whether the scan should be restricted to UCBs with 3-digit device numbers (3DIGIT) or should also include UCBs with 4-digit device numbers (ALL).

,NONBASE=NO
,NONBASE=YES
Specifies whether the scan should include nonbase exposures for a multiple-exposure device, supported on systems prior to MVS/ESA SP 5.2. NO specifies only the base exposure, and YES specifies all exposures.

Specifies whether the scan should include bound alias UCBs for a parallel access volume. NO specifies that bound alias UCBs will not be included. Yes specifies that bound alias UCBs will be included.

,UNBOUND_ALIAS=NO
,UNBOUND_ALIAS=YES
,UNBOUND_ALIAS=ONLY
Specifies whether the scan should include unbound alias UCBs.

YES	Include unbound alias UCBs
NO	Do not include unbound alias UCBs
ONLY	Include only unbound alias UCBs

Note: The UNBOUND_ALIAS function is intended for IOS use only.

,DEVCLASS=ALL
,DEVCLASS=CHAR
,DEVCLASS=COMM
,DEVCLASS=CTC
,DEVCLASS=DASD
,DEVCLASS=DISP
,DEVCLASS=TAPE
,DEVCLASS=UREC
Specifies the device class that is to be scanned:

ALL	Scans UCBs for all device classes
CHAR	Scans UCBs for character reader device class
COMM	Scans UCBs for communications device class

CTC	Scans UCBs for channel to channel device class
DASD	Scans UCBs for direct access device class
DISP	Scans UCBs for display device class
TAPE	Scans UCBs for tape device class
UREC	Scans UCBs for unit record device class

,DEVCID=*devcid addr*
Specifies the address of an 8-bit input field that contains the hexadecimal device class ID of the device class to be scanned.

If you specify DEVCID, only UCBs of the particular device class specified will be presented, and the DEVCLASS parameter is ignored.

,IOCTOKEN=*ioctoken addr*
,IOCTOKEN=NONE
Specifies the address of a 48-character storage area that contains the MVS I/O configuration token. The caller can obtain this token by issuing the IOCINFO macro. If the I/O configuration token that is current when UCBSCAN is invoked does not match the token whose address is supplied as input by *ioctoken addr*, the caller will be notified through a return code.

If the input IOCTOKEN (specified by *ioctoken addr*) is set to binary zeros, UCBSCAN will set IOCTOKEN to the current I/O configuration token at the start of the scan.

,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.
- **1**, 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 1

,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.

Return and Reason Codes

When control returns from USBSCAN, GPR 15 (and *retcode addr*, if you coded RETCODE) contains a return code and, for some return codes, GPR 0 (or *rsncode addr*, if you coded RSNCODE) contains a reason code.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** UCBSCAN completed successfully. **Action:** None.
04	01	**Meaning:** UCBSCAN processing ended. All UCBs that met the search criteria have been presented to the caller. The contents of UCBAREA are unchanged, and WORKAREA has been reset to binary zeros. **Action:** None.
08	01	**Meaning:** Program error. A caller in AR mode specified an ALET that was not valid. **Action:** Correct the ALET and reissue the macro. Possibly the caller wrote over an area in the parameter list; look for this error.
08	02	**Meaning:** Program error. An error occurred when the system tried to access the caller's parameter list. **Action:** Ensure that you have met the environmental requirements for the macro, and reissue the macro.
08	03	**Meaning:** Program error. An error occurred in referencing the caller-supplied area for the UCB copy; the area was specified in the UCBAREA parameter. **Action:** Correct the UCBAREA parameter.
08	04	**Meaning:** Program error. An error occurred in referencing the caller-supplied area for the UCB prefix extension segment data. This reason code is valid only for callers using the UCBPAREA parameter. **Action:** Correct the UCBPAREA parameter.
08	05	**Meaning:** Program error. An error occurred when the system referenced the caller-supplied area specified in the IOCTOKEN parameter. This reason code is valid only for callers using the IOCTOKEN parameter. **Action:** Correct the IOCTOKEN parameter.
08	08	**Meaning:** Program error. An error occurred in referencing the caller-supplied work area specified in the WORKAREA parameter. **Action:** Correct the WORKAREA parameter.
08	09	**Meaning:** Program error. An error occurred in referencing the caller-supplied CMXTAREA area. This reason code is valid only for callers using the CMXTAREA parameter. **Action:** Correct the CMXTAREA parameter.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
08	0B	**Meaning:** Program error. An error occurred in referencing the caller-supplied DCEAREA area. This reason code is valid only for callers using the DCEAREA parameter. **Action:** Correct the DCEAREA parameter.
08	0C	**Meaning:** Program error. The caller specified a volume serial number that is not valid. (Note that binary zeros are not considered valid.) This reason code is valid only for callers using the VOLSER parameter. **Action:** Correct the VOLSER parameter.
08	0D	**Meaning:** Program error. For the DCEAREA token, the caller specified a length that is negative, is zero, or exceeds 256 bytes. This reason code is valid only for callers using the DCELEN parameter. **Action:** Correct the DCELEN parameter.
0C	None	**Meaning:** Environmental error. The I/O configuration has changed, so that the I/O configuration token supplied through the IOCTOKEN parameter is not current. This return code is valid only for callers using the IOCTOKEN parameter. **Action:** Obtain the current I/O configuration token by issuing an IOCINFO macro or by setting the input IOCTOKEN parameter in the UCBINFO macro to zero. Start the scan from the beginning.
20	None	**Meaning:** System error. An unexpected error occurred. **Action:** Supply the return code to the appropriate IBM support personnel.

UCBSCAN COPY—List Form

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

Syntax

This macro is an alternative list form macro, and requires a different technique for using the list form as compared to the conventional list form macros. See “Alternative List Form Macros” on page 12 for further information.

The list form of the COPY function of the UCBSCAN macro is written as follows:

---

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

ƀ — One or more blanks must precede UCBSCAN.

UCBSCAN

b	One or more blanks must follow UCBSCAN.

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver:* 1
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 that standard form of the UCBSCAN macro with the following exceptions:

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

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

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

UCBSCAN COPY—Execute Form

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

Syntax

The execute form of the COPY function of the UCBSCAN macro is written as follows:

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

COPY

,WORKAREA=*workarea addr*	*workarea addr:* RX-type address or register (2) - (12).
,UCBAREA=*ucbarea addr*	*ucbarea addr:* RX-type address or register (2) - (12).
,CMXTAREA= *cmxtarea addr* ,CMXTAREA=NONE	*cmxtarea addr:* RX-type address or register (2) - (12). **Default:** NONE
,UCBPAREA=*ucbparea addr* ,UCBPAREA=NONE	*ucbparea addr:* RX-type address or register (2) - (12). **Default:** NONE
,DCEAREA= *dcearea addr* ,DCEAREA=NONE	*dcearea addr:* RX-type address or register (2) - (12). **Default:** NONE
,DCELEN=*length addr*	*length addr:* RS-type address or register (2) - (12). **Note:** DCELEN is valid only with DCEAREA and is required with DCEAREA.
,VOLSER=*volser addr* ,VOLSER=NONE	*volser addr:* RS-type address or register (2) - (12). **Default:** NONE
,DEVNCHAR=*devnchar addr*	*devnchar addr:* RS-type address or register (2) - (12).
,DEVN=*devn addr* ,DEVN=0	*devn addr:* RS-type address or register (2) - (12). **Default:** 0
,DYNAMIC=NO ,DYNAMIC=YES	**Default:** NO
,RANGE=3DIGIT ,RANGE=ALL	**Default:** 3DIGIT
,NONBASE=NO ,NONBASE=YES	**Default:** NO
,UNBOUND_ALIAS=NO ,UNBOUND_ALIAS=YES ,UNBOUND_ALIAS=ONLY	**Default:** NO
,DEVCLASS=ALL ,DEVCLASS=CHAR ,DEVCLASS=COMM ,DEVCLASS=CTC ,DEVCLASS=DASD ,DEVCLASS=DISP ,DEVCLASS=TAPE ,DEVCLASS=UREC	**Default:** ALL
,DEVCID=*devcid addr* ,DEVCID=0	*devcid addr:* RS-type address **Default:** 0
,IOCTOKEN=*ioctoken addr* ,IOCTOKEN=NONE	*ioctoken addr:* RX-type address or register (2) - (12). **Default:** NONE

,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default:** IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver:* 1
,RETCODE=*retcode addr*	*retcode addr:* RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr:* RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr:* RX-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

The parameters are explained under the standard form of the COPY function of the UCBSCAN macro with the following exceptions:

,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

Specifies the execute form of the UCBSCAN macro.

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

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

UPDTMPB — Update a Message Parameter Block for Substitution Data

Description

To build a message parameter block (MPB), you must issue both BLDMPB and UPDTMPB. BLDMPB initializes the MPB, and UPDTMPB adds one substitution token to the MPB each time you issue it. Issue UPDTMPB once for each substitution token in the message.

You can also use UPDTMPB to replace or change the value of a particular substitution token in an existing MPB. See *z/OS MVS Programming: Assembler Services Guide* for more information on using UPDTMPB.

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 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:	Not applicable

Programming Requirements

You must include the mapping macro CNLMMPB.

Restrictions

None.

Input Register Information

Before issuing the UPDTMPB 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 system
2-13	Unchanged
14	Used as a work register by 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 UPDTMPB macro is written as follows:

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

MPBPTR=*mpb addr*	*mpb addr*: RX-type address or register (2) - (12).
,MPBLEN=*mpb length addr*	*mpb length addr*: RX-type address or register (2) - (12).
,SUBOOFST=*new/changed blk offset addr*	*new/changed blk offset addr*: RX-type address or register (2) - (12).
,SUBCOFST=*existing blk offset addr*	*existing blk offset addr*: RX-type address or register (2) - 12).
,TOKEN=*token name addr*	*token name addr*: RX-type address or register (2) - (12).
,TOKLEN=*token length addr*	*token length addr*: RX-type address or register (2) - (12).
,TOKTYPE=*token type addr*	*token type addr*: RX-type address or register (2) - (12).
,SUBSDATA=*sub data addr*	*sub data addr*: RX-type address or register (2) - (12).
,SUBSLEN=*sub data length addr*	*sub data length addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

MPBPTR=*mpb addr*
specifies the address or a register containing the address of the MPB to be modified.

,MPBLEN=*mpb len addr*
specifies the address or a register containing the address of the length of the area addressed by MPBPTR.

,SUBOOFST=*new/changed blk offset addr*
specifies the address of the area or a register into which UPDTMPB returns the value of the offset from the start of the MPB to the new or changed substitution

block. A substitution block contains all the information that you need to format substitution data. It consists of a token field, token length, substitution length, token type, and substitution data.

,SUBCOFST=*existing blk offset addr*
specifies the address of the offset or a register containing the offset from the start of the MPB to the existing substitution block that UPDTMPB is to update. If you do not specify SUBCOFST, UPDTMBP will build a new substitution block.

,TOKEN=*token name addr*
specifies the address of the area or a register pointing to the area containing the substitution token name.

,TOKLEN=*token length addr*
specifies the address of the area or a register containing the length of the TOKEN field. If you do not specify TOKLEN, UPDTMPB uses, as a default, the length of the TOKEN field in the DSECT mapping. You must specify TOKLEN if you use register notation for the TOKEN keyword.

,TOKTYPE=*token type addr*
specifies the address of the area or a register containing the 1-byte token type. This field can have the following values and meanings:

Value	Meaning
0	text
1	date
2	time
3	day of week

,SUBSDATA=*sub data addr*
specifies the address of the area or a register pointing to the area containing the substitution data.

If TOKTYPE is 0, SUBSDATA can contain any text with a length defined SUBSLEN.

If TOKTYPE is 1, SUBSDATA must be eight bytes long and in the format *yyyymmdd*, where:

- *yyyy* is the year number, expressed as a 4-digit EBCDIC string in the range 0000 to 9999.
- *mm* is the month number, expressed as a 2-digit EBCDIC string in the range 01 to 12.
- *dd* is the day number, expressed as a 2-digit EBCDIC string in the range 01 to 31.

If TOKTYPE is 2, SUBSDATA must be twelve bytes long in the format *hhmmssdddddd*, where:

- *hh* is the hours in a 24-hour clock, expressed as a 2-digit EBCDIC string in the range 00 to 23.
- *mm* is the minutes, expressed as a 2-digit EBCDIC string in the range 00 to 59.
- *ss* is the seconds, expressed as a 2-digit EBCDIC string in the range 00 to 59. EBCDIC blanks are considered zeros.
- *dddddd* is the decimal seconds, expressed as a 6-digit EBCDIC string in the range 000000 to 999999. EBCDIC blanks are considered zeros.

If TOKTYPE is 3, SUBSDATA must be one byte long in the format *d*, where *d* is the day number, expressed as a 1-digit EBCDIC string in the range 1 to 7. The days are defined in parmlib member CNLcccxx. Day 1 is Sunday, 2 is Monday, and so on.

,SUBSLEN=*sub data length addr*
specifies the address of the area or a register pointing to the area containing the length of the substitution data. If you do not specify SUBSLEN, UPDTMPB uses, as a default, the length of the SUBSDATA field in the DSECT mapping. You must specify SUBSLEN if you use register notation for the SUBSDATA parameter.

Return and Reason Codes

When UPDTMPB completes, register 15 contains one of the following hexadecimal return codes:

Hexadecimal Return Code	Meaning
00	Processing completed successfully.
0C	Processing unsuccessful. See reason codes.

When UPDTMPB completes, register 0 contains one of the following hexadecimal reason codes:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00	00	Successful processing.
0C	33	There is insufficient storage in the MPB.
0C	35	The value for TOKLEN is either zero or negative.
0C	36	The value for SUBSLEN is negative.
0C	37	The TOKTYPE value is not valid.
0C	38	SUBCOFST is not valid.
0C	3B	The MPB acronym is not valid.

Example

Build and update an MPB for a message that contains substitution data for the third day of the week.

```
BLDMPBA  CSECT
BLDMPBA  AMODE  31
BLDMPBA  RMODE  ANY
         STM    14,12,12(13)
         BALR   12,0
         USING  *,12
         ST     13,SAVE+4
         LA     15,SAVE
         ST     15,8(13)
         LR     13,15
*********************************************************************
*        OBTAIN WORKING STORAGE AREA                                *
*********************************************************************
         GETMAIN RU,LV=STORLEN,SP=SP230
         LR    R4,R1
*
*********************************************************************
```

```
*        CREATE MPB HEADER SECTION                                     *
*********************************************************************
*
         BLDMPB MPBPTR=(R4),MPBLEN=MPBL,MSGID=MSGID,                    X
               MSGIDLEN=MIDLEN
*
*********************************************************************
*        ADD SUBSTITUTION DATA TO MPB                                  *
*********************************************************************
*
         LR    R2,R4
         A     R2,MPBL
         USING VARS,R2
*
         UPDTMPB MPBPTR=(R4),MPBLEN=MPBL,SUBOOFST=VARS,                 X
               TOKEN=TOKN,TOKLEN=TOKL,TOKTYPE=TOKT,                     X
               SUBSDATA=SDATA,SUBSLEN=SDATAL
*
*
*********************************************************************
*        FREE STORAGE AREA                                             *
*********************************************************************
*
         FREEMAIN RU,LV=STORLEN,SP=SP230,A=(4)
*
         L     13,SAVE+4
         LM    14,12,12(13)
         BR    14
*********************************************************************
MPBL     DC    A(MPBLEN)
MSGID    DC    CL10'MSGID2'
MIDLEN   DC    A(MIDL)
TOKN     DC    CL3'DAY'
TOKL     DC    F'3'
TOKT     DC    CL1'3'
SDATA    DC    CL1'3'
SDATAL   DC    A(SDL)
SAVE     DC    18F'0'
SP230    EQU   230
STORLEN  EQU   256
SDL      EQU   6
MIDL     EQU   6
MPBLEN   EQU   (MPBVDAT-MPB)+(MPBMID-MPBMSG)+(MPBSUB-MPBSB)+MIDL+SDL
R1       EQU   1
R2       EQU   2
R4       EQU   4
*********************************************************************
         DSECT
         CNLMMPB
VARS     DSECT
VARSAREA DS    CL24
VARSLEN  EQU   *-VARS
         END BLDMPBA
```

VRADATA — Update Variable Recording Area Data

Description

The VRADATA macro copies service information into a variable recording area (VRA), usually the system diagnostic work area (SDWAVRA). This information can later be recorded in the LOGREC data set if software errors occur. (See the SETRP macro, RECORD=YES parameter description, for more information on recording the SDWA data area.) The information copied into the VRA using this macro is in a key, length, data format defined by the IHAVRA mapping macro. The key and length are one-byte fields; the data can vary in length. The IHAVRA mapping macro is shown in *z/OS MVS Data Areas, Vol 5 (SSAG-XTLST)* under VRAMAP.

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- or 64-bit
ASC mode:	Primary, secondary, 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

- If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before issuing VRADATA. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.
- You must include the IHASDWA mapping macro as a DSECT in your program if you accept the default for VRAINIT, VRACLEN, VRAMLEN, or if you specify VRAINIT=SDWAVRA. You must also place the address of the SDWA data area into the SDWAREG register (or default register 1) if you accept the default for any of these three parameters.
- You must include the IHAVRA mapping macro as a DSECT in your program. If you include the IHASDWA mapping macro, IHAVRA is automatically included.
- You can issue VRADATA more than once in a program, but you need to specify VRAINIT, VRACLEN, and VRAMLEN only once for a particular series of updates to the VRA.
- If you specify a key on the KEY parameter, but no data on the DATA parameter, the length field for the VRA entry (LEN parameter) is zero. You must be running in the key the SDWA was obtained in. Refer to *z/OS MVS Programming: Assembler Services Guide* for more information.

Restrictions

None.

Input Register Information

Before issuing the VRADATA macro, the AR-mode caller must ensure that the following GPRs contain the specified information.

Register	Contents

© Copyright IBM Corp. 1988, 2002

1	Address of the SDWA if you do not specify the SDWAREG parameter on this invocation or any previous invocation of the VRADATA macro; otherwise, the caller does not have to place any information into this register.
14	Address of the next available field in the VRA if you do not specify the VRAREG parameter on this invocation or any previous invocation of the VRADATA macro; otherwise, the caller does not have to place any information into this register.

Before issuing the VRADATA macro, the caller must ensure that the following ARs contain the specified information.

Register	Contents
1	ALET of the SDWA whose address is in GPR 1, only if you do not specify the SDWAREG parameter on this invocation or any previous invocation of the VRADATA macro; otherwise, the caller does not have to place any information into this register.
14	ALET of the next available space in the VRA whose address is in GPR 14 only if you do not specify the VRAREG parameter on this invocation or any previous invocation of the VRADATA macro; otherwise, the caller does not have to place any information into this register.

Output Register Information

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

Register	Contents
0-13	Unchanged
14	Address of the next available space in the VRA for the next invocation of VRADATA if you did not specify the VRAREG parameter on this invocation or any previous invocation; otherwise, unchanged.
15	Used as a work register if you did not specify the WORKREG parameter on this invocation or any previous invocation of the VRADATA macro; otherwise, unchanged.

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 VRADATA macro is written as follows:

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

VRAINIT=*vra addr*	*vra addr*: RX-type address, or the symbol 'SDWAVRA'. **Default**: address of SDWAVRA
,VRACLEN=*curr len addr* or *(curr len addr,0)* ,VRAMLEN=*max len addr*	*curr len addr*: RX-type address. **Default**: address of SDWAURAL. *max len addr*: RX-type address. **Default**: address of SDWAVRAL.
,KEY=*key nmbr*	*key nmbr*: Symbol or decimal digit.
,LENADDR=*data len addr* ,LEN=*data len value*	*data len addr*: RX-type address. *data len value*: Symbol or decimal digit. **Default**: length of DATA storage.
,DATA=*data addr*	*data addr*: RX-type address, or register (1) - (15).
,SDWAREG=*reg*	*reg*: Symbol or decimal digits 1-15. **Default**: 1
,VRAREG=*(reg,descr)*	*reg*: Symbol or decimal digits 1-15. **Default**: 14 *descr*: SET or NOTSET **Default**: NOTSET if VRAINIT is specified, otherwise SET.
,WORKREG=*reg*	*reg*: Symbol or decimal digits 1-15. **Default**: 15
,TYPE=LEN,TEST ,TYPE=LEN,NOTEST ,TYPE=LEN,NOT ,TYPE=NOLEN,TEST ,TYPE=NOLEN,NOTEST ,TYPE=NOLEN,NOT ,TYPE=NOL,TEST ,TYPE=NOL,NOTEST ,TYPE=NOL,NOT	**Default**: LEN, TEST

Parameters

The parameters are explained as follows:

VRAINIT=*vra addr*
Specifies the address of the variable recording area to be initialized and updated. The value in the register specified by the VRAREG parameter is also initialized unless VRAREG=(,SET) is specified. If VRAINIT=SDWAVRA is specified, the SDWA data area is also updated to indicate that the VRA contains hexadecimal data, and data in key-length-data format. If VRAINIT is not specified, VRAINIT=SDWAVRA is assumed. All subsequent VRADATA macros use the specified VRAINIT value until you specify another VRAINIT value.

,VRACLEN=*curr len addr*
Specifies the address of a one-byte field that contains the length of the current VRA. This value changes as information is added in the VRA. If you do not specify VRACLEN, you can obtain the current length of the VRA from the SDWAURAL field of the SDWA.

,VRACLEN=*(curr len addr, 0)*
Specifies that the area containing the length is to be zeroed.

All subsequent VRADATA macros use the specified VRACLEN value until you specify another VRACLEN value.

,VRAMLEN=*max len addr*
Specifies the address of a two-byte field that contains the maximum length of the VRA. If you do not specify VRAMLEN, the maximum length is obtained from SDWAVRAL.

All subsequent VRADATA macros use the specified VRAMLEN value until you specify another VRAMLEN value.

,KEY=*key number*
Specifies the key value to be placed in the VRAKEY field of the current VRA entry. The IHAVRA mapping macro (VRAMAP) defines the valid key values.

,LENADDR=*data len addr*
,LEN=*data len value*
Specifies the length of the data for the VRA entry. The maximum length is 255 bytes. Omit this parameter unless the DATA parameter is a register value or a displacement plus a register, or if the defined data length must be overridden because it is larger than 255 bytes. For bit string data, use this parameter to indicate how many bytes the bit string occupies. The data length field pointed to by LENADDR must be a two-byte area with the length right-justified in the area.

,DATA=*data addr*
Specifies the address of the data to be copied into the VRA. The data must correspond to the key specified by the KEY parameter. If you specify DATA, you must specify KEY. You must also specify LEN or LENADDR if DATA has a register value or if the data length is greater than 255 bytes.

,SDWAREG=*reg*
Specifies a register containing the address of the SDWA data area. You must place the address in this register before invoking VRADATA. The VRADATA macro preserves the contents of this register. If you do not specify SDWAREG, register 1 is the default.

,VRAREG=*(reg,descr)*
Specifies a register to contain the address of the next available field in the VRA and a description of whether or not the register value is already set (SET) or not set (NOTSET). If VRAINIT is specified, the default is NOTSET. If VRAINIT

is not specified, the default is SET. If you specify NOTSET or default to it, the system program places the address of the VRA plus the current length in the register before updating the VRA.

After updating the VRA, the system updates the register to point to the next available field in the VRA. If you do not specify VRAREG, register 14 is the default.

,WORKREG=*reg*
Specifies a work register. Each time you invoke the VRADATA macro, the contents of this register are destroyed. If you do not specify WORKREG, register 15 is the default.

,TYPE=LEN,TEST
,TYPE=LEN,NOTEST
,TYPE=LEN,NOT
,TYPE=NOLEN,TEST
,TYPE=NOLEN,NOTEST
,TYPE=NOLEN,NOT
,TYPE=NOL,TEST
,TYPE=NOL,NOTEST
,TYPE=NOL,NOT
Specifies whether (LEN) or not (NOLEN) you want the current length of the VRA stored in the VRALEN area and also specifies whether (TEST) or not (NOTEST) you want the VRA tested to see if it is full before adding the new entry. If you specify TEST, the current length of the VRA must already be in the VRACLEN area.

If you do not need to store the length or test to see if the new entry fits, specify NOLEN and NOTEST. These specifications considerably reduce the amount of code generated by the VRADATA macro. If you do not specify TYPE, the value LEN, TEST is the default.

ABEND Codes

None.

Return and Reason Codes

None.

Example 1

Initialize the SDWA data area to indicate that the VRA contains hexadecimal data, in key, length, data format. Also, move two pieces of data into the SDWAVRA, and indicate that no test of the length of the VRA is needed, (because the data fits in the VRA). The second request indicates that the length used is to be stored in the VRA current length field. The pieces of data are the IHAVRA mapping macro name and the contents of a control block.

```
VRADATA VRAINIT=SDWAVRA,KEY=VRACBM,DATA=MYCBNAME,       X
        TYPE=(NOLEN,NOTEST)
VRADATA KEY=VRACB,DATA=MYCB,TYPE=(LEN,NOTEST)
```

Example 2

Initialize a variable recording area that is not the SDWA. Move in a piece of data, specifying its length. (The piece of data is an ASID.)

```
VRADATA VRAINIT=LRBTUSR,VRACLEN=LRBTCLEN,               X
        VRAMLEN=LBRTMLEN
VRADATA KEY=VRAAID,DATA=(REGA),LEN=ASIDLEN
```

WAIT — Wait for One or More Events

Description

The WAIT macro informs the system that performance of the active task cannot continue until one or more specific events, each represented by a different event control block (ECB), have occurred. Bit 0 and bit 1 of each ECB must be set to zero before it is used. The caller must be enabled, unlocked, and in primary address space control (ASC) mode.

The system takes the following action:

- For each event that has already occurred (each ECB is already posted), the count of the number of events is decreased by one.
- If the number of events is zero by the time the last event control block is checked, control is returned to the instruction following the WAIT macro.
- If the number of events is not zero by the time the last ECB is checked, control is not returned to the issuing program until sufficient ECBs are posted to bring the number to zero. Control is then returned to the instruction following the WAIT macro.

For more information on how to use the WAIT macro to synchronize tasks, see *z/OS MVS Programming: Assembler Services Guide*.

Environment

The requirements for callers of WAIT are:

Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	One of the following: • 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 interruptions
Locks:	No locks held
Control parameters:	ECB and ECBLIST must be in the home address space.

Programming Requirements

None.

Restrictions

When using LINKAGE=SVC (the default), the caller cannot have an EUT FRR established.

Input Register Information

Before issuing the WAIT 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 general purpose registers (GPRs) contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	For LINKAGE=SYSTEM: Used as work registers by the system For LINKAGE=SVC: Unchanged

When control returns to the caller, the access registers (AR) 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 WAIT macro is written as follows:

---

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

---

event nmbr,	*event nmbr*: Symbol, decimal digit, or register (0) or (2) - (12). **Default**: 1 **Value range**: 0-255
ECB=*ecb addr*	*ecb addr*: RX-type address, or register (1) or (2) - (12).
ECBLIST=*ecb list addr*	*ecb list addr*: RX-type address, or register (1) or (2) - (12).
,LONG=NO ,LONG=YES	**Default**: LONG=NO
,LINKAGE=SVC ,LINKAGE=SYSTEM	**Default**: LINKAGE=SVC
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

event nmbr,
Specifies the number of events waiting to occur.

ECB=*ecb addr*
ECBLIST=*ecb list addr*
Specifies the address of an ECB on a fullword boundary or the address of a virtual storage area containing one or more consecutive fullwords on a fullword boundary. Each fullword contains the address of an ECB; the high order bit in the last fullword must be set to one to indicate the end of the list.

The ECB parameter is valid only if the number of events is specified as one or is omitted. The number of ECBs in the list specified by the ECBLIST form must be equal to or greater than the specified number of events.

If you specify ECBLIST, *ecb list addr* and all ECBs on the list must be in the home address space.

,LONG=NO
,LONG=YES
Specifies whether the task is entering a long wait (YES) or a regular wait (NO).

,LINKAGE=SVC
,LINKAGE=SYSTEM
Specifies whether POST is to be called through an SVC (LINKAGE=SVC) or not (LINKAGE=SYSTEM).

When the caller is not in cross memory mode (the primary, secondary, and home address spaces are the same) and no EUT FRR is established, use LINKAGE=SVC. With this parameter, linkage is through an SVC instruction.

When the caller is in cross memory mode (the primary, secondary, and home address spaces are not the same) or if an EUT FRR is established, use LINKAGE=SYSTEM. With this parameter, linkage is through a PC instruction. Note that the ECB must be in the home address space.

,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.

The RELATED parameter is available on macros that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE) and on macros that relate to previous occurrences of the same macros (for example, CHAP and ESTAE).

The RELATED parameter may be used, for example, as follows:

```
WAIT1   WAIT     1,ECB=ECB,RELATED=(RESUME1,
                 'WAIT FOR EVENT')
  .
  .
  .
RESUME1 POST     ECB,0,RELATED=(WAIT1,
                 'RESUME WAITER')
```

Note: Each of these macros will fit on one line when coded, so there is no need for a continuation indicator.

CAUTION:
A job step with all of its tasks in a WAIT condition is terminated upon expiration of the time limits that apply to it.

Example

You have previously initiated one or more activities to be completed asynchronously to your processing. As each activity was initiated, you set up an ECB in which bits 0 and 1 were set to zero. You now wish to suspend your task via the WAIT macro until a specified number of these activities have been completed.

Completion of each activity must be made known to the system via the POST macro. POST causes an addressed ECB to be marked complete. If completion of the event satisfies the requirements of an outstanding WAIT, the waiting task is marked ready and will be executed when its priority allows.

ABEND Codes

WAIT might abnormally terminate with one of the following abend codes:
- X'101'
- X'201'
- X'301'
- X'401'

These hexadecimal codes are described in *z/OS MVS System Codes*.

Return and Reason Codes

None.

Example 1

Wait for one event to occur (with a default count).

```
        WAIT  ECB=WAITECB
           .
           .
WAITECB DC    F'0'
```

Example 2

Wait for 2 events to occur.

```
           WAIT   2,ECBLIST=LISTECBS
           .
           .
LISTECBS   DC  A(ECB1)
           DC  A(ECB2)
           DC  A(X'80000000'+ECB3)
```

Example 3

Enter a long wait for a task.

```
           WAIT   1,ECBLIST=LISTECBS,LONG=YES
           .
           .
           .
```

```
LISTECBS   DC  A(ECB1)
           DC  A(ECB2)
           DC  X'80'
           DC  AL3(ECB3)
```

WTL — Write To Log

Description

Note: **IBM recommends** you use the WTO macro with the MCSFLAG=HRDCPY parameter instead of WTL, because WTO supplies more data than WTL.

The WTL macro causes a message to be written to the system log (SYSLOG) or the operations log (OPERLOG) log stream depending on which one of these logs, or both, is active.

Note: When a message is recorded in SYSLOG, the exact format of the output of the WTL macro varies depending on the job entry subsystem (JES2 or JES3) that is being used, the output class that is assigned to the log at system initialization, and whether DLOG is in effect for JES3. See the following for information on the format of logged messages:

- *z/OS MVS System Messages, Vol 1 (ABA-AOM)*
- *z/OS MVS System Messages, Vol 2 (ARC-ASA)*
- *z/OS MVS System Messages, Vol 3 (ASB-BPX)*
- *z/OS MVS System Messages, Vol 4 (CBD-DMO)*
- *z/OS MVS System Messages, Vol 5 (EDG-GFS)*
- *z/OS MVS System Messages, Vol 6 (GOS-IEA)*
- *z/OS MVS System Messages, Vol 7 (IEB-IEE)*
- *z/OS MVS System Messages, Vol 8 (IEF-IGD)*
- *z/OS MVS System Messages, Vol 9 (IGF-IWM)*
- *z/OS MVS System Messages, Vol 10 (IXC-IZP)*

z/OS JES3 Commands also contains information on the format of logged messages.

Environment

The requirements for the caller are:

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

Programming Requirements

None.

Restrictions

Message text cannot exceed 126 characters. If the message text exceeds 126 characters, truncation occurs at the last embedded blank before the 126th character; when there are no embedded blanks, truncation occurs after the 126th character.

© Copyright IBM Corp. 1988, 2002

Input Register Information

Before issuing the WTL 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
1-14	Unchanged
15	Return code

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

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

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

Performance Implications

None.

Syntax

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

---

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

'msg'	*msg*: Up to 126 characters.

Parameters

The parameter is explained as follows:

'msg'
Specifies the message to be written to the system log and/or the operations log. The message must be enclosed in apostrophes, which will not appear in the

system log. The message can include any character that can be used in a C-type (character) DC statement, and is assembled as a variable-length record. See “Timing and Communication” in *z/OS MVS Programming: Assembler Services Guide* for a list of the printable EBCDIC characters passed to display devices or printers.

ABEND Codes

None.

Return and Reason Codes

When the WTL macro returns control to your program, GPR 15 contains a return code and GPR 0 contains a reason code. WTL issues a return code (either 00 or 04), with multiple reason codes for each. The return codes indicate the following:

- 00 - WTL wrote the message to the system log, the operations log, or both.
- 04 - WTL could not write the message to either the system log or the operations log.

Return Code	Reason Code	Meaning and Action
0	None	**Meaning**: WTL processing completed successfully. The system logged the message in SYSLOG, and, if OPERLOG was requested, the system logged the message in OPERLOG. **Action**: None.
0	04	**Meaning**: WTL processing completed successfully. The message was logged in the operations log (OPERLOG log stream). The system log was not active. **Action**: If you want the message logged in the system log, start the system log and rerun the program.
0	08	**Meaning**: WTL processing completed, but the message was only logged in the operations log because the WTL system log buffers are full. **Action**: Do one of the following, if you want subsequent messages logged in the system log: • Enter a CONTROL M,LOGLIM command to change the allocated number of WTL system log buffers dynamically. • Change the LOGLIM value, specifying the number of WTL system log buffers on the INIT statement in the CONSOLxx parmlib member. This value will take effect at the next IPL.
0	0C	**Meaning**: WTL processing completed, but the message was only logged in the system log because the operations log was not active. **Action**: If you want the message logged in the operations log, start the operations log and rerun the program. This will also place the message in the system log.

Return Code	Reason Code	Meaning and Action
0	10	**Meaning**: WTL processing completed, but the message was only logged in the system log. The message was not logged in the OPERLOG log stream because of a storage problem. **Action**: If you want the message logged in the operations log, retry the request. This will also place the message in the system log. If the problem persists, contact the IBM Support Center. Provide the return and reason code.
04	04	**Meaning**: System error. WTL processing was not successful. Recovery could not be established. **Action**: Retry the request. If the problem persists, record the return and reason code and supply them to the appropriate IBM support personnel.
04	08	**Meaning**: Environmental error. The system log and the operations log are not active. **Action**: Start the logs and rerun your program.
04	0C	**Meaning**: Environmental error. The WTL limit has been reached. **Action**: Do one of the following: 1. Retry the request when the shortage is relieved. 2. Issue a CONTROL M,LOGLIM command to change the allocated number of WTL SYSLOG buffers. 3. Change the LOGLIM value on the INIT statement in the CONSOLxx member of SYS1.PARMLIB. This new value will take effect at the next IPL. **Note:** If the problem is persistent, you might want to perform step 2 first and step 3 at the next IPL.
04	10	**Meaning**: System error. An internal error occurred. The system issues message IEE390I. **Action**: Contact the IBM Support Center. Provide the return and reason code.
04	14	**Meaning**: System error. The system encountered a (VSM) error. The system issues message IEE390I. **Action**: Contact the IBM Support Center. Provide the return and reason code.
04	18	**Meaning**: Environmental error. The message was not logged in either the system log or the operations log, because neither log is active. **Action**: Do one of the following: • If you want to log the message in the operations log, start the operations log with the VARY OPERLOG,HARDCPY command and rerun the program. • If you want the message logged in the system log, start the system log (SYSLOG) with the VARY SYSLOG,HARDCPY command and rerun the program.

Return Code	Reason Code	Meaning and Action
04	1C	**Meaning**: Environmental error. The message was not logged in the system log, as requested, because the WTL limit has been reached. The operation log was not active at the time, so the message was not logged there either. **Action**: To log the message in the system log, do the following: • Issue a CONTROL M,LOGLIM command to change the allocated number of WTL SYSLOG buffers. • Change the LOGLIM value on the INIT statement in the CONSOLxx member of SYS1.PARMLIB. This new value will take effect at the next initialization. • Retry the request when the storage shortage has been relieved. If the problem persists, issue the CONTROL M,LOGLIM command first, and change the LOGLIM value in CONSOLxx at your next IPL. To log the message in the operations log, start the operations log and rerun the program.
04	20	**Meaning**: Environmental error. The message was not logged in the operations log, as requested, because of storage problems. The system log was not active. **Action**: To log the message in the operations log, retry the request. If the problem persists, contact the IBM Support Center, providing the return and reason codes. To log the message in the system log also, start the system log and rerun the program.
04	24	**Meaning**: Environmental error. The message was not logged in the system log because the WTL limit has been reached, and was not logged in the operation log because of storage problems. **Action**: To log the message in the operations log, retry the request. If the problem persists, contact the IBM Support Center, providing the return and reason codes.

Example 1

Write a message to the system log.

```
WTL   'THIS IS THE STANDARD FORMAT FOR THE WTL MACRO'
```

Example 2

Write a message constructed in the list form of WTL.

```
WTL   MF=(E,(R2))
```

WTL—List Form

The list form of the WTL macro is used to construct a control program parameter list. The message parameter must be provided in the list form of the macro.

Syntax

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

name	*name*: Symbol. Begin *name* in column 1.
b	One or more blanks must precede WTL.
WTL	
b	One or more blanks must follow WTL.
'msg'	*msg*: Up to 126 characters.
,MF=L	

Parameters

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

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

WTL — Execute Form

The execute form of the WTL macro uses a remote control program parameter list. The parameter list can be generated by the list form of WTL. You cannot modify the message in the execute form.

Syntax

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

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

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 WTL macro with the following exception:

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

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

WTO — Write to Operator

Description

The WTO macro allows you to write messages to one or more operator consoles. See *z/OS MVS Programming: Assembler Services Guide* for more information on using WTO.

Environment

Requirements for the caller are:

Minimum authorization:	Problem state and any PSW key.
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

Be aware of the following when coding the WTO macro:

- If you code MCSFLAG=REG0, you must use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeroes). However, IBM recommends using the CONSID parameter rather than register 0.
- If you are not coding MCSFLAG=REGO, you should clear register zero.
- If the list and execute forms of the WTO macro are in separate modules, both modules must be assembled or compiled with the same level of WTO.
- If the execute form of the macro specifies TEXT=*(text addr)*, CART, KEY, TOKEN, CONSID, or CONSNAME, then the list form, to ensure that the parameter list is generated correctly, must specify the same parameters without data. For example:

  ```
  WTO 'USR001I FOR SPECIAL REQUESTS CONTACT SYSTEM SUPPORT',CONSID=,MF=L
  ```

  If you specify parameter values on the list form, the system issues an MNOTE and ignores the data.
- For any WTO parameters that allow a register specification, the value must be right-justified in the register.
- If you specify the TEXT keyword for a multi-line WTO, you must code its parameters in the following way:
  - On the list form, omit *text addr* for each line, but include *line type*. If you specify *text addr*, the system ignores the data and issues an MNOTE.
  - On the execute form, omit *line type* for each line, but include *text addr*.
- When using any parameter with an address, the data being referenced must be accessible by the caller issuing the WTO.

Restrictions

- You can issue a WTO of up to 10 lines. A WTO over 10 lines produces a return code of 04. The return code indicates that only 10 lines will be processed and the rest are ignored.

© Copyright IBM Corp. 1988, 2002

- The caller cannot have an EUT FRR established.

Input Register Information

Before issuing the WTO 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, or MCSFLAG=REG0 has been coded.

Output Register Information

When control returns to the caller, the output registers contain the following values:

Register	Contents
0	Used as a work register by the system unless WTO returns code X'20' in register 15. In that case, register 0 contains the number of active WTO buffers for the issuer's address space.
1	Message identification number if the WTO macro completed normally (you can use this number to delete the message when it is no longer needed); otherwise, used as a work register by the system.
2-13	Unchanged.
14	Used as a work register by the system.
15	Return code.

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

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

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

Performance Implications

Users who cannot wait because of a WTO buffer shortage should use the MCSFLAG=BUSYEXIT parameter and then take appropriate action on the busy return.

Syntax

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

---

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

'msg'
(*'text')*
(*'text',line type*)
TEXT=(*text addr*)
TEXT=(*text addr,line type*)
TEXT=*((text addr,line type),...(text addr,line type))*

msg: Up to 126 characters.
text: Up to 126 characters.

text addr: RX-type address or register (2) - (12).
Note: If you code *'msg'* or *('text'...)*, it must be the first positional parameter.

The permissible *line types*, text lengths, and maximum numbers of each line type are shown below:

```
line type    text        maximum number
C            34 char     1 C type
L            70 char     2 L type
D            70 char     10 D type
DE           70 char     1 DE type
             or
E            None        1 E type
```

The maximum total number of lines that can be coded in one instruction is 10.

,ROUTCDE=(*routing code*)

routing code: Decimal digit from 1 to 28. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.

,MCSFLAG=(*flag name*)

flag name: Any combination of the following, separated by commas:

REG0	HRDCPY
RESP	REPLY
NOTIME	BRDCST
CMD	BUSYEXIT

,DESC=(*descriptor code*)

descriptor code: Decimal number from 1 to 13. The *descriptor code* is one or more codes, separated by commas.

,CART=*cmd/resp token*

cmd/resp token: RX-type address or register (2) - (12).

,KEY=*key*

key: RX-type address or register (2) - (12).

,TOKEN=*token*

token: RX-type address or register (2) - (12).

,CONSID=*console id*
,CONSNAME=*console name*

console id: RX-type address or register (2) - (12).
console name: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

'msg'
(*'text'***)**
(*'text',line type***)**
TEXT=*(text addr)*
TEXT=*(text addr,line type)*
TEXT=((*text addr,line type),...(text addr,line type***))**

Specifies the message or multiple-line message to be written to one or more operator consoles.

The parameter *'msg'* is used to write a single-line message to the operator. In the format, the message must be enclosed in apostrophes, which do not appear on the console. To have apostrophes appear in the message text, use two apostrophes to get one to appear. For example, ''Message Off'' would appear on a display as 'Message Off'. The message text can include any character that can be used in a character (C-type) DC instruction. When a program issues a WTO macro, the system translates the text; only standard printable EBCDIC characters are passed to MCS-managed display devices. The EBCDIC characters that can be displayed are listed in *z/OS MVS Programming: Assembler Services Guide*. All other characters are replaced by blanks. Unless the console has dual-case capability, lowercase characters are displayed or printed as uppercase characters.

The message is assembled as a variable-length record. The parameters TEXT=(*text addr*) and TEXT=(*text addr,line type*) represent a 4-byte address of a message to be displayed. The message consists of a 2-byte message length followed by the message text. The 2-byte message length describes the length of the message text only. There are no boundary requirements.

The parameters (*'text'*) and (*text addr,line type*) are used to write a multiple-line message to the operator. The text is one line of the multiple-line message. Inline text consists of a character string enclosed in apostrophes (which do not appear on the operator console). Any character valid in a C-type DC instruction can be coded. The maximum number of characters depends on which line type is specified. The message can be up to ten lines long; the system truncates the message at the end of the tenth line. The ten-line limit does not include the control line (message IEE9321I), as explained under line type C below.

Notes:

1. If the parameter (*'text'*) is coded without repetition, for example, (*'text'*), the message appears as a single-line message.
2. All lines of a multiple-line WTO must be consistently specified with the message text or the TEXT keyword. When coding the TEXT keyword for a multiple-line message:
   - You can specify a maximum of 10 lines.
   - Do not exceed the 70-character limit for the macro parameter value.
3. For a multiple-line message, you must clear the three high-order bytes of register 0.

The line type defines the type of information contained in the "text" field of each line of the message:

C Indicates that the "text" parameter is the text to be contained in the control line of the message. The control line normally contains a message title. C may only be coded for the first line of a multiple-line message. If this parameter is omitted and descriptor code 9 is coded, the system generates a control line (message IEE932I) containing only a message identification number. The control line remains static while you scroll through all the lines of a multiple-line message displayed on an MCS console (provided that the message is displayed in an out-of-line display area). Control lines are optional.

L Indicates that the "text" parameter is a label line. Label lines contain message heading information; they remain static while you scroll through all the lines of a multiple-line message displayed on an MCS console (provided that the message is displayed in an out-of-line display area). Label lines are optional. If coded, lines must either immediately follow the control line, or another label line or be the first

line of the multiple-line message if there is no control line. Only two label lines may be coded per message.

D Indicates that the “text” parameter contains the information to be conveyed to the operator by the multiple-line message. While you scroll through all lines of a multiple-line message displayed on an MCS console, the data lines are paged.

DE Indicates that the “text” parameter contains the last line of information to be passed to the operator. Specify DE on the last line of text of the WTO. If there is no text on the last line, specify E.

E Indicates that the previous line of text was the last line of text to be passed to the operator. The “text” parameter, if any, coded with a line type of E is ignored. If the last line has text, specify DE.

,ROUTCDE=*(routing code)*

Specifies the routing code or codes to be assigned to the message.

The routing codes are:

Message Routing Code	Definition
1	Master console action
2	Master console information
3	Tape pool
4	Direct access pool
5	Tape library
6	Disk library
7	Unit record pool
8	Teleprocessing control
9	System security
10	System error/maintenance/system programmer information
11	Programmer information
12	Emulators
13-20	Reserved for customer use
21-28	Reserved for IBM- or customer-defined subsystem use

If you omit the ROUTCDE, DESC, and CONSID or CONSNAME parameters, the system uses the routing code specified on the ROUTCODE parameter on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.

Note: Routing codes 1, 2, 3, 4, 7, 8, and 10 cause hard copy of the message when display consoles are used, or more than one console is active. All other routing codes may go to hard copy as a PARMLIB option or as a result of a VARY HARDCPY command.

,MCSFLAG=(*flag name***)**

Specifies one or more flag names whose meanings are shown below:

Table 42. MCSFLAG Flag Names for WTO Macro

Flag Name	Meaning
REG0	Queue the message to the console whose console ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) to identify the console to receive the message. However, **IBM recommends** that you use the CONSID parameter instead of register 0.

Table 42. MCSFLAG Flag Names for WTO Macro (continued)

Flag Name	Meaning
RESP	The WTO is an immediate command response.
REPLY	This WTO is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.
HRDCPY	Queue the message for hard copy only.
NOTIME	Do not append time to the message.
CMD	The WTO is a recording of a system command issued for hardcopy log purposes.
BUSYEXIT	If there are no message or console buffers for either MCS or JES3, or there is a JES3 WTO staging area excess, the WTO is terminated with a x'20' return code and a reason code, in register 0, equal to the number of active WTO buffers for the issuer's address space. If BUSYEXIT is not specified, the WTO will go into a wait state if WTO buffers are not available.

,DESC=(*descriptor code***)**

Specifies the message descriptor code or codes to be assigned to the message. Descriptor codes 1 through 6, 11 and descriptor code 12 are mutually exclusive. Codes 7 through 10, and 13, can be assigned in combination with any other code.

The descriptor codes are:

Message Descriptor Code	Definition
1	System failure
2	Immediate action required
3	Eventual action required
4	System status
5	Immediate command response
6	Job status
7	Retain action message for until the job step terminates
8	Out-of-line message
9	Operator request
10	Dynamic status displays
11	Critical eventual action requested
12	Important information messages
13	Message previously automated

Action messages may have an * sign or @ sign displayed before the first character of the message. The * sign indicates that the WTO was issued by an authorized program. The @ sign indicates that the WTO was issued by an unauthorized program. These action messages will cause the audible alarm to sound on operator consoles so-equipped.

All WTO messages with descriptor codes of 1, 2, or 11 are action messages that have an @ sign printed before the first character. This indicates a need for operator action.

The system holds messages with descriptor codes 1, 2, 3, or 11 until you delete them. When you no longer need messages with descriptor codes 1, 2, 3, or 11,

you should delete those messages using the DOM macro. If messages with descriptor codes 1, 2, 3, or 11 also have descriptor code 7, the system deletes them automatically at job step. The system adds descriptor code 7 to all messages with descriptor code 1 or 2.

On operator consoles that support color, descriptor codes determine the color in which a message should be displayed. The colors used are described in *z/OS MVS System Commands*.

The message processing facility (MPF) can suppress messages. For MPF to suppress messages, the hardcopy log must be active. The suppressed messages do not appear on any console; they do appear on the hardcopy log.

,CART=*cmd/resp token*
Specifies an 8-character input field containing a command and response token to be associated with this message. The command and response token is used to associate user information with a command and its command response. You can supply any value as a command and response token. When you specify this parameter in the list form, code it as CART= with nothing after the equal sign.

,KEY=*key*
Specifies an input field containing an 8-byte key to be associated with this message. The key must be EBCDIC if used with the MVS DISPLAY R command for retrieval purposes, but it must not be '*'. If a register is used, it contains the address of the key. When you specify this parameter in the list form, code it as KEY= with nothing after the equal sign.

,TOKEN=*token*
Specifies an input field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token must be unique within an address space and can be any value. When you specify this parameter in the list form, code it as TOKEN= with nothing after the equal sign.

Note: When you code the TOKEN parameter using a register, the register must contain the token itself, rather than the address of the token.

,CONSID=*console id*
Specifies a 4-byte field containing the ID of the console to receive a message. Use this ID in place of a console ID in register 0. If you specify a 4-byte console ID, or if you specify a console ID for an extended MCS console, you must use CONSID instead of register 0. If you specify a 1-byte console ID, you must right-justify it and pad to the left with zeros. To view a list of valid console IDs, issue the DISPLAY CONSOLES command.

Notes:

1. If you code the CONSID parameter using a register, the register must contain the console ID itself, rather than the address of the console ID.
2. When you code CONSID on the list form of WTO, code it as CONSID= with nothing after the equal sign.
3. Do not use both CONSID and register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSID parameter to an existing invocation of WTO.
4. CONSID is mutually exclusive with the CONSNAME parameter.

,CONSNAME=*console name*
Specifies an 8-byte field containing a 2- through 8-character name, left-justified

and padded with blanks, of the console to receive a message. When you specify this parameter in the list form, code it as CONSNAME= with nothing after the equal sign.

This parameter is mutually exclusive with the CONSID parameter. Do not use CONSNAME to pass a console name, together with register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSNAME parameter to an existing invocation of WTO.

ABEND Codes

WTO might abnormally terminate with abend code X'D23'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

When the WTO macro returns control to your program, GPR 15 contains one of the following return codes:

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Processing completed successfully. **Action**: None.
04	**Meaning**: Program error. One of the following occurred: • The number of lines passed was 0. The request was ignored. • The number of lines passed was greater than 10. Only 10 lines were processed. • The message text length for a line was less than 1. All lines up to the error line were processed. **Action**: • Make sure your text is properly referenced. If you are using the TEXT parameter, make sure it is pointing to valid data. • Make sure your message text contains no more than 10 lines. • Make sure your message text is defined correctly. If you are using the TEXT parameter, make sure the first two bytes of data in the area pointed to by the TEXT parameter value contain the length of the message text. In all cases, correct the problem and retry the request.
0C	**Meaning**: Program error. A line type is not valid. An end has been forced at the point of the error unless the first line is an E line, in which case the request was ignored. All messages up to this one in the multiline request were processed. **Action**: Determine if a line type value on your multiline message was not syntactically correct. Correct the problem and retry the request.
20	**Meaning**: Environmental error. WTO processing has been terminated because it would have caused a wait state, and BUSYEXIT was specified. Register 0 contains the number of active WTO buffers for the issuer's address space. **Action**: Retry the request when the buffer storage constraint has been relieved.

Hexadecimal Return Code	Meaning and Action
30	**Meaning**: Environmental error. For routing code 11, the required resource was not available and the request was ignored. For any other routing code, the request was processed. **Action**: Retry the request when the resource you need is available.

Example 1

Issue a WTO with routing codes 1 and 10, descriptor code 2.

```
 WTO     'USR001I CRITICAL RESOURCE SHORTAGE DETECTED',          X
         ROUTCDE=(1,10),                                         X
         DESC=(2)
```

Example 2

Issue a WTO using the TEXT parameter. The message is to be sent to a console whose ID is contained in register 5 as a command response. A command and response token is also defined for this message. This example assumes a console ID was stored in field SAVECNID and a cart in SAVECART prior to issuing the WTO.

```
R0      EQU     0
R4      EQU     4
R5      EQU     5
        .
        .
        .
        LA      R4,MYMSG                ADDRESS OF MESSAGE AREA
        L       R5,SAVECNID             CONSOLE ID
        XR      R0,R0                   CLEAR REGISTER 0
        WTO     TEXT=(R4),CONSID=(R5),CART=SAVECART,                    X
                DESC=(5)
        .
        .
        .
MYMSG    DC     AL2(L'CATTXT)
CATTXT   DC     C'USR100I PROCESSING COMPLETE, NO ERRORS.'
SAVECART DS     CL8
SAVECNID DS     F
```

Example 3

Issue a multiline message using the TEXT parameter. This is an important information message which is not to be sent to the hardcopy log.

```
R0      EQU     0
        .
        .
        .
        XR      R0,R0                   CLEAR REG0 BEFORE MULTILINE
        WTO     TEXT=((MESSAG1,D),(MESSAG2,D),(MESSAG3,DE)),            X
                DESC=(7,12)
        .
        .
        .
MESSAG1 DC   AL2(L'MSG1TXT)
MSG1TXT DC   C'USR005I ALL JOBS REQUIRING MORE THAN 2 TAPES MUST BE RUNX
               ON THIRD SHIFT'
MESSAG2 DC   AL2(L'MSG2TXT)
MSG2TXT DC   C'JOBS REQUIRING 2 TAPES MAY BE RUN ON SECOND SHIFT'
MESSAG3 DC   AL2(L'MSG3TXT)
MSG3TXT DC   C'OR ON FIRST SHIFT WITH THE OPERATOR'S PERMISSION.'
```

WTO—List Form

Use the list form of the WTO 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 WTO macro is written as follows:

---

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

'msg'
(*'text'*)
(*'text',line type*)
TEXT=
TEXT=*((,line type),(,line type),...(,line type))*
	msg: Up to 126 characters.
	text: Up to 126 characters.
	Notes:
	1. If you code *'msg'* or *('text'...)*, it must be the first parameter you code.
	2. For a single-line WTO, the parameter value is not required on TEXT for the list form. Code only TEXT=. Then code TEXT=(*text addr*) on the execute form.
	
	The permissible *line types*, text lengths, and maximum numbers of each line type are shown below:
	
	`line type  text  maximum number`
	`C  34 char  1 C type`
	`L  70 char  2 L type`
	`D  70 char  10 D type`
	`DE  70 char  1 DE type`
	`  or`
	`E  None  1 E type`
	
	The maximum total number of lines that can be coded in one instruction is 10.

,ROUTCDE=(*routing code*)	*routing code*: Decimal digit from 1 to 28. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.
,MCSFLAG=(*flag name*)	*flag name*: Any combination of the following, separated by commas: REG0 HRDCPY RESP REPLY NOTIME BRDCST CMD BUSYEXIT
,DESC=(*descriptor code*)	*descriptor code*: Decimal digit from 1 to 13. The *descriptor code* is one or more codes, separated by commas.

,CART=	Parameter value not required for list form. Code only CART=. If you code CART on the list form of WTO, you must code CART on the execute form.
,KEY=	Parameter value not required for list form. Code only KEY=. If you code KEY on the list form of WTO, you must code KEY on the execute form.
,TOKEN=	Parameter value not required for list form. Code only TOKEN=. If you code TOKEN on the list form of WTO, you must code TOKEN on the execute form.
,CONSID= ,CONSNAME=	Parameter value not required for list form. Code only CONSID= or CONSNAME=. If you code CONSID (or CONSNAME) on the list form of WTO, you must code CONSID (or CONSNAME) on the execute form.

,MF=L

Parameters

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

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

Example

Set up the list form of a WTO, and send an immediate action message to the master console.

```
MYLIST   WTO   'USR001I CRITICAL RESOURCE SHORTAGE DETECTED',          X
               ROUTCDE=(1,10),                                          X
               DESC=(2),CONSID=,MF=L
```

WTO—Execute Form

Use the execute form of the WTO 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 message cannot be modified on the execute form of the macro if you code inline text (*'msg'* or *('text'...)*) on the list form.

Syntax

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

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

WTO

b	One or more blanks must follow WTO.

TEXT=*(text addr)* TEXT=*((text addr,),(text addr,),...(text addr,))*	*text addr*: RX-type address or register (2) - (12). **Notes:** 1. If you code TEXT=*(text addr)* on the execute form of WTO, you must code TEXT= on the list form. 2. If you specify inline text on the list form (*'msg'* or (*'text'...*)), do not code the TEXT keyword on the execute form.
,CART=*cmd/resp token*	*cmd/resp token*: RX-type address or register (2) - (12). If you code CART on the execute form of WTO, you must code CART on the list form.
,KEY=*key*	*key*: RX-type address or register (2) - (12). If you code KEY on the execute form of WTO, you must code KEY on the list form.
,TOKEN=*token*	*token*: RX-type address or register (2) - (12). If you code TOKEN on the execute form of WTO, you must code TOKEN on the list form.
,CONSID=*console id* ,CONSNAME=*console name*	*console id*: RX-type address or register (2) - (12). *console name*: RX-type address or register (2) - (12). If you code CONSID (or CONSNAME) on the execute form of WTO, you must code CONSID (or CONSNAME) on the list form.
,MF=(E,*list addr*)	*list addr*: RX-type address, or register (1) - (12).

Parameters

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

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

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

Example 1

Write a message with a prebuilt parameter list pointed to by register 1.

```
WTO     MF=(E,(1))
```

Example 2

Issue a WTO whose list form is defined at label MYLIST, and is pointed to by register 2. Send the WTO to the console with an ID of 1, pointed to by register 4.

```
R2      EQU   2
R4      EQU   4
        .
        .
        .
```

```
         LA    R2,MYLIST                ADDRESS OF PARAMETER LIST
         L     R4,MYCONID               CONSOLE ID
         WTO   MF=(E,(R2)),CONSID=R4
         .
         .
         .
MYCONID  DC    F'1'
```

WTOR — Write to Operator with Reply

Description

The WTOR macro causes a message requiring a reply to be written to one or more operator consoles and the hardcopy log. The macro also provides the information required by the system to return the reply to the issuing program. See *z/OS MVS Programming: Assembler Services Guide* for more information on using the WTOR macro.

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

Environment

Requirements for the caller are:

Minimum authorization:	Problem state and any PSW key
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

Be aware of the following when coding the WTOR macro:

- If the list and execute forms of the WTOR macro are in separate modules, both modules must be assembled or compiled with the same level of WTOR.
- The parameter list for WTOR must begin on a fullword boundary.
- If the execute form of the macro specifies RPLYISUR, CART, CONSID, CONSNAME, KEY, or TOKEN, the list form, to ensure that the parameter list is generated correctly, must specify the same parameters without data. If you specify parameter values on the list form, the system issues an MNOTE and ignores the data.
- For any WTOR parameters that allow a register specification, the value must be right-justified in the register.

Restrictions

- The WTOR macro can issue only single-line messages.
- The caller cannot have an EUT FRR established.

Input Register Information

Before issuing the WTOR 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	Used as a work register by the system.
1	Message identification number if the WTOR macro completed normally (you can use this number to delete the message when it is no longer needed); otherwise, used as a work register by the system.
2-13	Unchanged.
14	Used as a work register by the system.
15	Return code.

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

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14-15	Used as work registers by the system

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

Performance Implications

None.

Syntax

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

---

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

'msg',reply addr,reply length,ecb addr TEXT=*(text addr,reply addr,reply length,ecb addr)*	*msg*: Up to 122 characters. *text addr*: RX-type address or register (2) - (12). *reply addr*: A-type address, or register (2) - (12). *reply length*: Symbol, decimal number, or register (2) - (12). The minimum length is 1; the maximum length is 119. *ecb addr*: A-type address, or register (2) - (12).
,ROUTCDE=(*routing code*)	*routing code*: Decimal digit from 1 to 28. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.
,MCSFLAG=(*flag name*)	*flag name*: Any combination of the following, separated by commas: REG0 HRDCPY RESP REPLY NOTIME BRDCST CMD

,DESC=(*descriptor code*)	*descriptor code*: Decimal number 7 or 13. If you code both 7 and 13, separate them with commas.
,MSGTYP=*(msg type)*	*msg type*: Any of the following: N SESS,JOBNAMES Y SESS,STATUS SESS JOBNAMES,STATUS JOBNAMES SESS,JOBNAMES,STATUS STATUS
,RPLYISUR=*reply console*	*reply console*: RX-type address or register (2) - (12).
,CART=*cmd/resp token*	*cmd/resp token*: RX-type address or register (2) - (12).
,CONSID=*console id*	*console id*: RX-type address or register (2) - (12).
,CONSNAME=*console name*	*console name*: RX-type address or register (2) - (12).
,KEY=*key*	*key*: RX-type address or register (2) - (12).
,TOKEN=*token*	*token*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

'msg',reply addr,reply length,ecb addr
TEXT=(*text addr,reply addr,reply length,ecb addr***)**

'msg' is used to write the message to the operator. The message must be enclosed in apostrophes, which do not appear on the console. It can include any character that can be used in a character (C-type) DC instruction. When a program issues a WTOR macro, the system translates the text; only standard printable EBCDIC characters are passed to the display devices. All other characters are replaced by blanks. A list of these EBCDIC characters is provided in *z/OS MVS Programming: Assembler Services Guide*. Unless the console has dual-case capability, lowercase characters are converted to uppercase by the display station or printer and displayed or printed as uppercase characters.

The message is assembled as a variable-length record. *text addr* contains an address that points to a message to be displayed. The message contains a 2-byte text field length followed by the text. The 2-byte message length describes the length of the message text only. There are no boundary requirements.

Note: All WTOR messages are action messages. An indicator is printed before the first character of an action message to indicate a need for operator action. Action messages will cause the audible alarm to sound on operator consoles so-equipped.

reply addr specifies the address in virtual storage of the area into which the system is to place the reply. The reply is left-justified at this address.

reply length specifies the length, in bytes, of the reply message.

ecb addr specifies the address of the event control block (ECB) to be used by the system to indicate the completion of the reply and the ID of the replying console. The value of the ECB data must point to a fullword boundary. The ECB should be zeroed before the WTOR issued. After the system receives the reply, the ECB appears as follows:

Offset	Length(bytes)	Contents
0	1	Completion code
1	2	Not part of the intended programming interface
3	1	1-byte console ID in binary

Note: Use RPLYISUR to obtain the 4-byte console id and console name of the console issuing the reply.

,ROUTCDE=(*routing code***)**
Specifies the routing code or codes to be assigned to the message.

The routing codes are:

Message Routing Code	Definition
1	Master console action
2	Master console information
3	Tape pool
4	Direct access pool
5	Tape library
6	Disk library
7	Unit record pool
8	Teleprocessing control
9	System security
10	System error/maintenance/system programmer information
11	Programmer information
12	Emulators
13-20	Reserved for customer use
21-28	Reserved for IBM- or customer-defined subsystem use

If you omit the ROUTCDE, and CONSID or CONSNAME parameters, the system uses the routing code specified on the ROUTCODE parameter on the DEFAULT statement in the CONSOLxx member of SYS1.PARMLIB.

,MCSFLAG=*(flag name)*
Specifies one or more flag names whose meanings are shown below:

Table 43. MCSFLAG Flag Names for WTOR Macro

Flag Name	Meaning
REG0	Queue the message to the console whose console ID is passed in register 0. You can use register 0 to pass a 1-byte console ID (right-justified and padded to the left with zeros) to identify the console to receive the message. However, IBM recommends you use the CONSID parameter instead of register 0.
RESP	The WTOR is an immediate command response.
REPLY	This is a reply to a WTOR.
BRDCST	Broadcast the message to all active consoles.

Table 43. MCSFLAG Flag Names for WTOR Macro (continued)

Flag Name	Meaning
HRDCPY	Queue the message for hard copy only.
NOTIME	Do not append time to the message.
CMD	The WTOR is a recording of a system command issued for hardcopy log purposes.

,DESC=(*descriptor code***)**

Specifies the message descriptor code or codes to be assigned to the message. Valid descriptor codes for the WTOR macro are:

7 Retain action message for life-of-task

13 Message previously automated

All WTOR messages are action messages that have an @ sign displayed before the first character. This indicates a need for operator action.

The system adds descriptor code 7 to all WTOR messages. The system holds all WTOR messages until one of the following events occurs:

- The system deletes the WTOR message when the reply is received.
- You delete the WTOR message using the DOM macro. You should delete any unanswered WTOR messages that are no longer current.
- The system deletes the WTOR message at task termination.

The message processing facility (MPF) can suppress messages. For MPF to suppress messages, the hardcopy log must be active. The suppressed messages do not appear on any console; they do appear on the hardcopy log.

,MSGTYP=*(msg type)*

Specifies how the message is to be routed to consoles on which the MONITOR command is active. If you specify anything other than MSGTYP=N, which is the default, your message is routed according to your specification on MSGTYP, and the ROUTCDE parameter is ignored.

For SESS, JOBNAMES, or STATUS, the message is to be routed to the console that issued the MONITOR SESS, MONITOR JOBNAMES, or MONITOR STATUS command, respectively. When the message type is identified by the operating system, the message is routed to only those consoles that requested the information.

For Y or N, the message type describes what functions (MONITOR SESS, MONITOR JOBNAMES, and MONITOR STATUS) are desired. N, or omission of the MSGTYP parameter, indicates that the message is to be routed as specified in the ROUTCDE parameter. Y creates an area in the WTO parameter list in which you can set message type information if you are coding a WTOR without any of the following parameters:

- KEY
- TOKEN
- CONSID
- CONSNAME
- TEXT
- RPLYISUR
- CART
- LINKAGE
- SYNCH

IBM recommends that you do not use MSGTYP=Y.

,RPLYISUR =*reply console*
Specifies a 12-byte field where the system will place the 8-byte console name and the 4-byte console ID of the console through which the operator replies to this message. When you specify this keyword in the list form, code it as RPLYISUR= with nothing after the equal sign.

,CART=*cmd/resp token*
Specifies an 8-byte field containing a command and response token to be associated with this message. The command and response token is used to associate user information with a command and its command response. When you specify this keyword in the list form, code it as CART= with nothing after the equal sign.

,CONSID=*console id*
Specifies a 4-byte field containing the ID of the console to receive a message. To view a list of valid console IDs, issue the DISPLAY CONSOLES command. Use this ID in place of a console ID in register 0. If you specify a 4-byte console ID, or if you specify a console ID for an extended MCS console, you must use CONSID instead of register 0. If you specify a 1-byte console ID, you must right-justify it and pad to the left with zeros.

Notes:

1. If you code the CONSID parameter using a register, the register must contain the console ID itself, rather than the address of the console ID.
2. When you code CONSID on the list form of WTOR, code it as CONSID= with nothing after the equal sign.
3. Do not use both CONSID and register 0 to pass a console ID, because the results are unpredictable.
4. CONSID is mutually exclusive with the CONSNAME parameter.

,CONSNAME=*console name*
Specifies an 8-byte field containing a 2- through 8- character name, left-justified and padded with blanks, of the console to receive a message. This parameter is mutually exclusive with the CONSID parameter. When you specify this keyword in the list form, code it as CONSNAME= with nothing after the equal sign. Do not use CONSNAME to pass a console name, together with register 0 to pass a console ID, because the results are unpredictable. Be sure to clear the low-order byte of register 0 if you add the CONSNAME parameter to an existing invocation of WTOR.

,KEY=*key*
Specifies a field containing an 8-byte key to be associated with this message. The key must be EBCDIC if used with the MVS DISPLAY R command for retrieval purposes, but it must not be '*'. The key must be left-justified and padded on the right with blanks. If a register is used, it contains the address of the key. When this keyword is specified in the list form, it must be coded as KEY= with nothing after the equal sign.

,TOKEN=*token*
Specifies a field containing a 4-byte token to be associated with this message. This field is used to identify a group of messages that can be deleted by a DOM macro that includes TOKEN. The token must be unique within an address space and can be any value. When you specify this keyword on the list form, code it as TOKEN= with nothing after the equal sign.

Note: When you code the TOKEN parameter using a register, the register must contain the token itself, rather than the address of the token.

ABEND Codes

WTOR might abnormally terminate with abend code X'D23'. See *z/OS MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

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

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Processing completed successfully. **Action**: None. Be sure to delete the request by issuing the DOM macro.
04	**Meaning**: Program error. One of the following occurred: • The number of lines passed was 0; the request was ignored. • The message text length for a line was less than 1; all lines up to the error line were processed. **Action**: Correct the problem and retry the request. If you used the TEXT parameter, verify the data pointed to by the parameter.

Example 1

Issue a WTOR to a console whose ID is in register 4.

```
         WTOR  'USR902A REPLY YES OR NO TO CONTINUE.',REPLY,L8,REPECB, X
               CONSID=(R4),RPLYISUR=CONINFO
         .
         .
         .
R4       EQU   4
L8       EQU   8
REPLY    DS    CL8
REPECB   DS    F
CONINFO  DS    CL12
```

Example 2

Issue a WTOR with the TEXT parameter. The message is to go to a specific console whose name is in field TOCON.

```
R4       EQU   4
LENG72   EQU   72
         .
         .
         .
         LA    R4,CATMSG
         WTOR  TEXT=(CATMSG,REPAREA,LENG72,IDSECB),                    X
               CONSNAME=TOCON,                                         X
               RPLYISUR=IDSAREA
         .
         .
         .
CATMSG   DC    AL2(L'REP99)
REP99    DC    C'USR999A ENTER LIST OF USERIDS.'
```

```
TOCON    DC    CL8'ALTCON  '
REPAREA  DS    CL72
IDSECB   DS    F
IDSAREA  DS    CL12
```

Example 3

Issue a WTOR using the TEXT parameter with the list and execute forms of the macro. The console ID to which the message is to be queued is assumed to be in field MYCONID. On the TEXT parameter for the execute form, commas mark the positions of *reply addr* and *ecb addr*; for the list form, a comma marks the position of *reply length*.

```
R12      EQU   12
C50      EQU   50                      LENGTH OF REPLY AREA
         USING *,R12
         .
         .
         .
         WTOR  MF=(E,M2,EXTENDED),TEXT=(MESSAGE,,C50,),CONSID=MYCONID, X
               RPLYISUR=MYCONAR
         .
         .
         .
M2       DS    0H
         WTOR  TEXT=(,RAREA,,MYECB),CONSID=,ROUTCDE=(2),RPLYISUR=,MF=L
MYCONID  DS    F
RAREA    DS    CL50
MYECB    DS    F
MYCONAR  DS    CL12
MESSAGE  DC    AL2(L'MTEXT)
MTEXT    DC    C'USR930A REQUEST IS AMBIGUOUS. RESPECIFY DEVICE.'
         END
```

WTOR—List Form

Use the list form of the WTOR 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 message parameter must be provided in the list form.

Syntax

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

---

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

'msg',reply addr,reply length,ecb addr
TEXT=(*,reply addr,reply length,ecb addr*)

msg: Up to 122 characters.
reply addr: A-type address.
reply length: Symbol or decimal number.
The minimum length is 1; the maximum length is 119.
ecb addr: A-type address.

Notes:

1. If you code *'msg',reply addr,reply length,ecb addr*, it must be the first parameter you code.
2. If you do not code *reply addr* on the list form of WTOR, mark its position with a comma, and code *reply addr* on the execute form. The same is true for *reply length* and *ecb addr*.

,ROUTCDE=(*routing code*)

routing code: Decimal digit from 1 to 28. The *routing code* is one or more codes, separated by commas, or a hyphen to indicate a range.

,MCSFLAG=(*flag name*)

flag name: Any combination of the following, separated by commas:

REG0	HRDCPY
RESP	REPLY
NOTIME	BRDCST
CMD	

,DESC=(*descriptor code*)

descriptor code: Decimal number 7 or 13. If you code both 7 and 13, separate them with commas.

,RPLYISUR=

Parameter value not required for list form. Code only RPLYISUR=. If you code RPLYISUR on the list form of WTOR, you must code RPLYISUR on the execute form.

,CART=

Parameter value not required for list form. Code only CART=. If you code CART on the list form of WTOR, you must code CART on the execute form.

,CONSID=
,CONSNAME=

Parameter value not required for list form. Code only CONSID= or CONSNAME=.
If you code CONSID (or CONSNAME) on the list form of WTOR, you must code CONSID (or CONSNAME) on the execute form.

,KEY=

Parameter value not required for list form. Code only KEY=. If you code KEY on the list form of WTOR, you must code KEY on the execute form.

,TOKEN=

Parameter value not required for list form. Code only TOKEN=. If you code TOKEN on the list form of WTOR, you must code TOKEN on the execute form.

,MF=L

Parameters

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

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

WTOR—Execute Form

Use the execute form of the WTOR 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 message cannot be modified on the execute form of the macro if you code inline text (*'msg'...*) on the list form.

Syntax

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

---

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

,reply addr,reply length,ecb addr TEXT=*(text addr,reply addr,reply length,ecb addr)*	*reply addr*: RX-type address, or register (2) - (12). *reply length*: Symbol, decimal number, or register 2-12. The minimum length is 1; the maximum length is 119. *ecb addr*: RX-type address, or register (2) - (12). *text addr*: RX-type address or register (2) - (12). **Notes:** 1. If you code *reply addr,reply length,ecb addr*, it must be the first parameter you code and must be preceded by a comma. 2. If you specify inline text on the list form (*'msg'...*), do not code the TEXT keyword on the execute form. 3. If you do not code *reply addr* on the execute form of WTOR, mark its position with a comma, and code *reply addr* on the list form. The same is true for *reply length* and *ecb addr*.
,RPLYISUR=*reply console*	*reply console*: RX-type address or register (2) - (12). If you code RPLYISUR on the execute form of WTOR, you must code RPLYISUR on the list form.
,CART=*cmd/resp token*	*cmd/resp token*: RX-type address or register (2) - (12). If you code CART on the execute form of WTOR, you must code CART on the list form.
,CONSID=*console id* ,CONSNAME=*console name*	*console id*: RX-type address or register (2) - (12). *console name*: RX-type address or register (2) - (12). If you code CONSID (or CONSNAME) on the execute form of WTOR, you must code CONSID (or CONSNAME) on the list form.
,KEY=*key*	*key*: RX-type address or register (2) - (12). If you code KEY on the execute form of WTOR, you must code KEY on the list form.

,TOKEN=*token*	*token*: RX-type address or register (2) - (12). If you code TOKEN on the execute form of WTOR, you must code TOKEN on the list form.
,MF=(E,*list addr*) ,MF=(E,*list addr*,EXTENDED)	*list addr*: RX-type address, or register (1) - (12).

Parameters

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

,reply addr,reply length,ecb addr
If you code *reply addr,reply length,ecb addr*, it must be the first parameter you code and must be preceded by a comma.

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

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

If you specify *reply addr*, *reply length*, *ecb addr*, or RPLYISUR on the execute form of WTOR, together with one or more of the following parameters, you must specify EXTENDED for the system to generate the parameter list correctly:
KEY
TOKEN
CONSID
CONSNAME
TEXT
RPLYISUR
CART
ROUTCDE (greater than 16)

XCTL and XCTLX — Pass Control to a Program in Another Load Module

Description

The XCTL macro passes control to a specified entry name in a load module; the entry name must be a member name, an alias in a directory of a partitioned data set, or have been specified in an IDENTIFY macro. The system brings the load module (called the **target module**) containing the entry name into storage if a usable copy is not already available. Control passes from the program that issues the XCTL or XCTLX (called the **XCTL issuer**) to the target module; control does not return to the XCTL issuer. Rather, control returns to the program that caused the XCTL issuer to run. The use count for the XCTL issuer's load module is decremented by 1. If the use count becomes zero, the system deletes the XCTLX issuer's module and reassigns that storage.

Descriptions of the XCTL and XCTLX macro in this book are:

- The standard form of the XCTL macro, which includes general information about the XCTL and XCTLX macros with specific information about the XCTL macro. The syntax of the XCTL macro and all XCTL parameters are described.
- The standard form of the XCTLX macro, which presents information specific to the XCTLX macro. The topic explains the syntax of the XCTLX macro and the parameters that are valid only on XCTLX.
- The list form of the XCTL and XCTLX macros.
- The execute form of the XCTL and XCTLX macros.

The XCTL or XCTLX issuer can pass data to the target module in register 1 in several ways:

- Using XCTL without LSEARCH and PARAM, placing the data directly in register 1. This choice is not available to the caller in AR mode.
- Using the execute form of the macro, placing the address of the data on the MF parameter. For this choice, the issuer might have used the CALL macro to build a user parameter list.
- Using the execute form of XCTL or XCTLX, specifying the location or locations of the data on the PARAM parameter. XCTL or XCTLX builds a list of the addresses (a user parameter list) at the location you specify on the MF parameter.

The data passed to the target module must not reside within the XCTL issuer's module; if the system deletes the XCTL issuer's module, any data in that module is not available. For more help in understanding passing parameters with XCTL and XCTLX, see "Examples of Passing Data to the Target Module" on page 799.

The target module gets control in the residency mode and addressing mode established by the link-edit. If XCTL=YES was specified on the ESTAE or ESTAEX macro that set up recovery for the XCTL issuer, then the ESTAE-type recovery routine covers the target module also.

The target module must return to the program that caused the XCTL issuer to run. According to linkage conventions, the target module is responsible for restoring the status of the program that originally caused the XCTL issuer to run. The status includes the contents of registers 2 through 14, as well as other information that is expected by the program that caused the XCTL issuer to run, such as:

- The program interruption control area (PICA)

© Copyright IBM Corp. 1988, 2002

- The program mask.

The system abnormally terminates the task under either of the following conditions:

- The system cannot locate the entry point that is to receive control
- The XCTL issuer added entries to the linkage stack, and did not remove those entries prior to issuing the XCTL.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=SASN=HASN
AMODE:	24-bit or 31-bit for XCTL; 24- or 31- or 64-bit for XCTLX
ASC mode:	Primary or access register
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must reside in the primary address space
User parameters:	Must reside in the primary address space

Syntax

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

---

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

(*reg1*), (*reg1,reg2*),	*reg1* and *reg2*: Decimal digits in the order 2 through 12.
EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address or register (2) - (12).
DE=*list entry addr*	*list entry addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: LSEARCH=NO

Parameters

The parameters are explained as follows:

(reg1),
(reg1,reg2),
Specifies the register or range of registers to be restored before the target routine gets control from the save area at the address contained in register 13. Note that the registers must be specified as decimal numbers; forms like "(R2,R12)" are not accepted.

EP=*entry name*
EPLOC=*entry name addr*
DE=*list entry addr*
Specifies the entry name, the address of the entry name, or the address of a 62-byte list entry for the entry name that was constructed using the BLDL macro. If EPLOC is coded, the name must be padded to eight bytes, if necessary.

The system ignores the information you specify on the DE parameter if the parameter does one or both of the following:

- Specifies an entry in an authorized library (that is, defined in IEAAPFxx member of parmlib)
- Requests access to a program or library that is controlled by the system authorization facility (SAF)

Instead, the system uses the BLDL macro to construct a new list entry containing the DE information.

Note: When you use the DE parameter with the XCTL macro, DE specifies the address of a list that was created by a BLDL macro. BLDL and XCTL must be issued from the same task; otherwise, the system might terminate the program with an abend code of 106 and a return code of 15. Therefore, do not issue an ATTACH or a DETACH macro between issuances of the BLDL and the XCTL macros.

,DCB=*dcb addr*
Specifies the address of the opened data control block for the partitioned data set containing the entry name described above. This parameter must indicate the same DCB used in the BLDL mentioned above. The DCB must not be defined in the XCTL issuer.

If the DCB parameter is omitted or if DCB=0 is specified when the XCTL macro is issued by the job step task, the data sets referred to by either the STEPLIB or JOBLIB DD statement are first searched for the entry name. If the entry name is not found, the link library is searched.

If the DCB parameter is omitted or if DCB=0 is specified when the XCTL macro is issued by a subtask, the data sets associated with one or more data control blocks referred to by the TASKLIB operand of previous ATTACH macros in the subtasking chain are first searched for the entry point name. If the entry point name is not found, the search is continued as if the XCTL had been issued by the job step task.

Note: The DCB must reside in 24-bit addressable storage.

,LSEARCH=NO

,LSEARCH=YES
Specifies whether (YES) or not (NO) you want the search limited to the job pack area and the first library in the normal search sequence.

Note: When you use LSEARCH on XCTL, the system does not pass the contents of register 1 to the target module, unless you specify MF=(E,(1)) on the execute form.

Return and Reason Codes

None.

Example

Pass control through the address of the entry name (XCTLEP), and have registers 2 through 12 restored.

```
XCTL   (2,12),EPLOC=XCTLEP
```

XCTLX — Pass Control to a Program in Another Load Module

The XCTLX macro performs the same function as XCTL: it causes control to pass to a specified entry name in another load module, the target module. XCTLX is intended for use by programs running in access register (AR) mode. Programs running in primary mode can also use XCTLX.

If your program runs in AR mode, before you issue the XCTLX macro, issue the SYSSTATE ASCENV=AR macro to tell the XCTLX macro to generate code appropriate for AR mode.

Syntax

The XCTLX macro is written as follows:

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

(*reg1*), (*reg1,reg2*),	*reg1 and reg2*: Decimal digits in the order 2 through 12.
EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address or register (2) - (12).
DE=*list entry addr*	*list entry addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: LSEARCH=NO

Parameters

The parameters are described under the syntax of the standard form of the XCTL macro.

XCTL and XCTLX—List Form

Two parameter lists are used on XCTL or XCTLX: a control parameter list and an optional user parameter list. The list form uses only the control parameter list. The execute form builds a user parameter list and passes it to the target module.

Syntax

The list form of the XCTL or XCTLX macro is written as follows:

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

EP=*entry name,*	*entry name*: Symbol.
EPLOC=*entry name addr,*	*entry name addr*: A-type addresses.
DE=*list entry addr,*	*list entry addr*: A-type address.
,DCB=*dcb addr,*	*dcb addr*: A-type address.
,LSEARCH=NO, ,LSEARCH=YES,	**Default**: LSEARCH=NO
,SF=L	

Parameters

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

,SF=L
Specifies the list form of the XCTL or XCTLX macro.

Note: If you code LSEARCH in either the list or execute form of the macro, you must code it in both.

XCTL—Execute Form

Two parameter lists are available in the XCTL macro: a control parameter list and an optional user parameter list. The control parameter list can be either inline or remote (that is, in an area you specifically obtained); the user parameter list **must** be remote.

Syntax

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

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

(*reg1*), (*reg1,reg2*),	*reg1* and *reg2*: Decimal digits or RX-type addresses, and in the order 2 through 12.
EP=*entry name,* EPLOC=*entry name addr,* DE=*list entry addr,*	*entry name*: Symbol. *entry name addr*: RX-type address or register (2) - (12). *list entry addr*: RX-type address, or register (2) - (12).
,DCB=*dcb addr,*	*dcb addr*: RX-type address, or register (2) - (12).
,PARAM=(*parm*), ,PARAM=(*parm*),VL=1,	*parm*: RX-type address, or register (2) - (12). *parm* is one or more addresses, separated by commas. For example, PARAM=(*parm,parm,parm*)
,LSEARCH=NO, ,LSEARCH=YES,	**Default**: LSEARCH=NO
,MF=(E,*user area*) ,SF=(E,*ctrl area*) ,MF=(E,*user area*),SF=(E,*ctrl area*)	*user area*: RX-type address, or register (1) or (2) - (12). *ctrl area*: RX-type address, or register (2) - (12) or (15).

Parameters

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

PARAM=(*parm*)
PARAM=(*parm*),VL=1
Specifies one or more parameters to be passed to the target module. XCTL builds the user parameter list consisting of a fullword address for each

parameter in the order specified, placed at the location designated by MF=(E,*user area*). When the target module gets control, register 1 contains the address of the location designated by *user area*.

Use VL=1 if you are passing the target module a variable number of parameters. VL=1 causes the high-order bit of the last address parameter to be set to 1; the target module can check the last bit to find the end of the list.

LSEARCH=NO
LSEARCH=YES
Specifies whether (YES) or not (NO) you want the search limited to the job pack area and to the first library in the normal search sequence.

Notes:

1. Do not use register 1 to pass parameters to the target module unless you use XCTL and omit both LSEARCH and PARAM.
2. If you code LSEARCH in either the list or execute form of the macro, you must code it in both.

,MF=(E,*user area***)**
,SF=(E,*ctrl area***)**
,MF=(E,*user area***),SF=(E,***ctrl area***)**
Specifies the execute form of the XCTL macro.

Use MF=(E,*user area*) to specify the address of data you want the target module to receive in register 1. If you specify PARAM, MF=(E,*user area*) is required and identifies the remote location where you want XCTL to build the parameter list.

Use SF=(E,*ctrl area*) to point to a remote control parameter list. If you do not specify SF, XCTL builds the control parameter list inline.

XCTLX—Execute Form

Two parameter lists are available in the XCTLX macro: a control parameter list and an optional user parameter list. The control parameter list can be either inline or remote (that is, in an area you specifically obtained); the user parameter list **must** be remote.

Syntax

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

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

(*reg1*), (*reg1,reg2*),	*reg1* and *reg2*: Decimal digits or RX-type addresses, and in the order 2 through 12.

EP=*entry name,*	*entry name*: Symbol.
EPLOC=*entry name addr,*	*entry name addr*: RX-type address or register (2) - (12).
DE=*list entry addr,*	*list entry addr*: RX-type address, or register (2) - (12).
,DCB=*dcb addr,*	*dcb addr*: RX-type address, or register (2) - (12).
,PARAM=(*parm*), ,PARAM=(*parm*),VL=1,	*parm*: RX-type address, or register (2) - (12). *parm* is one or more addresses, separated by commas. For example, PARAM=(*parm,parm,parm*)
,LSEARCH=NO, ,LSEARCH=YES,	**Default**: LSEARCH=NO
,PLIST4=YES, ,PLIST4=NO,	**Default**: None.
,PLIST8=YES, ,PLIST8=NO,	**Default**: None.
,MF=(E,*user area*) ,SF=(E,*ctrl area*) ,MF=(E,*user area*),SF=(E,*ctrl area*)	*user area*: RX-type address, or register (1) or (2) - (12). *ctrl area*: RX-type address, or register (2) - (12) or (15).

Parameters

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

PARAM=(*parm*)
PARAM=(*parm*),VL=1

Specifies one or more parameters to be passed to the target module. XCTLX builds the user parameter list consisting of a fullword address for each parameter in the order specified, placed at the location designated by MF=(E,*user area*). When the target module gets control, register 1 contains the address of the location designated by *user area*.

If the caller is in AR mode, XCTLX builds the user parameter list so that the addresses passed to the target module are in the first half of the parameter list, and their corresponding ALETs are in the last half of the list. See Figure 1 on page 5 for more information about the format of the user parameter list.

Use VL=1 if you are passing the target module a variable number of parameters. VL=1 causes the high-order bit of the last address parameter to be set to 1; the target module can check the last bit to find the end of the list.

LSEARCH=NO
LSEARCH=YES

Specifies whether (YES) or not (NO) you want the search limited to the job pack area and to the first library in the normal search sequence.

Note: If you code LSEARCH in either the list or execute form of the macro, you must code it in both.

,PLIST4=YES

,PLIST4=NO

,PLIST8=YES
,PLIST8=NO

Defines the size of the parameter list entries for a parameter list to be built by XCTLX based on the PARAM keyword.

PLIST4 and PLIST8 cannot be specified together. If neither is specified, the default is:

- If running AMODE 64, PLIST8=YES
- If not running AMODE 64, PLIST4=YES

If running AMODE 64 and PLIST4=YES is specified, the system builds a 4-bytes-per-entry parameter list just as it would if the program were running AMODE 24 or AMODE 31 and did not specify PLIST4 or PLIST8.

If running AMODE 24 or AMODE 31 and PLIST8 is specified, the system builds an 8-bytes-per-entry parameter list just as it would if the program were running AMODE 64 and did not specify PLIST4 or PLIST8.

,MF=(E,*user area*)
,SF=(E,*ctrl area*)
,MF=(E,*user area*),SF=(E,*ctrl area*)

Specifies the execute form of the XCTL macro.

Use MF=(E,*user area*) to specify the address of data you want the target module to receive in register 1. If you specify PARAM, MF=(E,*user area*) is required and identifies the remote location where you want XCTLX to build the parameter list.

Use SF=(E,*ctrl area*) to point to a remote control parameter list. If you do not specify SF, XCTLX builds the control parameter list inline.

Examples of Passing Data to the Target Module

These examples all perform the following function: pass control using the address of the entry name (XCTLEP), have registers 2 through 12 restored, and have the target module receive data in register 1. The control parameter list is inline.

Example 1

An XCTL issuer (not in AR mode) wants to pass a 6-byte token to the target module. The issuer puts the token into register 1 and issues the macro.

```
XCTL   (2,12),EPLOC=XCTLEP
```

When the target module receives control, register 1 contains the token.

Example 2

An XCTL issuer (not in AR mode) wants to pass data that resides at the location ADDRDATA.

```
XCTL   (2,12),EPLOC=XCTLEP,MF=(E,ADDRDATA)
```

When the target module receives control, register 1 contains the address of ADDRDATA.

Example 3

An XCTLX issuer (in primary or AR mode) wants to pass an address of a parameter list that was built by the CALL macro. The parameter list resides at the location PARM1. Additionally, the issuer wants to limit the search for the target module.

```
XCTLX  (2,12),EPLOC=XCTLEP,LSEARCH=YES,MF=(E,PARM1)
```

When the target module receives control, register 1 contains the address of PARM1.

Example 4

An XCTLX issuer (in primary or AR mode) wants to pass a parameter list consisting of the addresses of three parameters. The issuer wants XCTLX to build a user parameter list at the address contained in register 3, and then pass this address to the target module. The three parameters are DATA1, DATA2, and DATA3.

```
XCTLX  (2,12),EPLOC=XCTLEP,PARAM=(DATA1,DATA2,DATA3),MF=(E,(3))
```

When the target module receives control, register 1 contains the address of the user parameter list that contains the fullword addresses of DATA1, DATA2, and DATA3, in that order.

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 all 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:

- AD/Cycle
- AFP
- CICS
- DFSMS/MVS
- ESA/390
- Hiperspace
- IBM
- IBMLink
- MVS/DFP
- MVS/ESA
- MVS/SP
- MVS/XA
- OS/390
- RACF
- Resource Link
- SP
- SP1
- SP2
- Sysplex Timer
- 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
    STIMER macro 571
    SYNCH 631
    WTOR macro 779
    XCTL macro 791

A

accessibility 801
action message 770, 781
addressing mode and the services 2
ALET qualification
  of parameters 3
AMODE (addressing mode)
  changing
    using the LINK macro 447
AR () mode
  description 3
ASC (address space control) mode
  defining 3

B

branch macro
  converting to relative branch 81

C

callable service
  coding 15
central storage
  loading virtual storage 469
coding the callable services 15
coding the macros 12
completion of an event
  signalling 487
continuation line 15
control
  passing to another load module 447

D

data sharing with IARVSERV macro 29
device measurement block index
  obtaining 205
disability 801
documents, licensed xii

E

ECB (event control block)
  setting 487
ETR (external time reference)
  checking for TOD clock synchronization with 567
event
  signalling completion 487

I

I/O configuration token
  obtaining 205
IARR2V macro 23
IARV64 macro 41
IARVSERV macro
  data sharing 29
IDENTIFY macro 73
IEAARR macro 77
IEABRC macro 81
IEAFP macro 83
IEAINTKN macro 87
IEALSQRY macro 89
IEANTCR callable service 93
IEANTDL macro 99
IEANTRT callable service 103
IEAVAPE callable service 127
IEAVDPE callable service 131
IEAVPSE callable service 135
IEAVRLS callable service 141
IEAVRPI callable service 145
IEAVTPE callable service 151
IEAVXFR callable service 155
IEFDDSRV macro 161
IEFPRMLB macro 169
IEFSSI macro 197
incident token
  building 87
IOCINFO macro 205
IOS (input/output supervisor)
  building control unit entry 217
  obtaining information 211
IOSCHPD macro 211
IOSCUMOD macro 217
ITTUINIT service 221
ITTUTERM service 225
ITTUWRIT service 229
ITZEVENT macro 233
ITZQUERY macro 243
IXGBRWSE macro 249
IXGCONN macro 293
IXGDELET macro 315
IXGIMPRT macro 331
IXGINVNT macro 345
IXGOFFLD macro 411
IXGQUERY macro 421
IXGUPDAT macro 431
IXGWRITE macro 431

K

keyboard 801

© Copyright IBM Corp. 1988, 2002

L

licensed documents xii
LINK and LINKX macros 447
linkage stack
  query macro 89
LOAD macro 457
load module
  adding an entry name 73
  bringing into virtual storage 457
  passing control 447
  responsibility count 457
Logrec Data Set
  symptom record entries from SYMREC macro 623
LookAt message retrieval tool xii
LSEXPAND macro 465

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
multiple timer
  setting 577

N

Notices 803

P

paging service
  PGLOAD macro 469
  PGOUT macro 473
  PGRLSE macro 477
  PGSER macro 481
PGLOAD macro 469
PGOUT macro 473
PGRLSE macro 477
PGSER macro 481
POST macro 487
process symptom record 623
program object
  bringing into virtual storage 457

Q

QRYLANG macro 491

R

REFPAT macro 497
RESERVE macro 505
responsibility count
  for a load module 457
RETURN macro 515

S

SAVE macro 517
service
  ALET qualification 3
  summary 16
services
  addressing mode 2
  ASC mode
    defining 3
  using 1
SETRP macro 519
shared DASD
  reserve a device 505
sharing storage with IARVSERV macro 29
shortcut keys 801
SNAP and SNAPX macros 525
specify program interruption exit
  See SPIE macro
SPIE macro 539
SPLEVEL macro 545
STAE macro 549
STATUS macro 555
STCKCONV macro 561
STCKSYNC macro 567
STIMER macro 571
STIMERM macro 577
STORAGE macro 591
subtask
  changing status 555
symptom record 623
SYMRBLD macro 605
SYMREC macro 623
SYNCH and SYNCHX macros 631
synchronous exit 634
SYSEVENT macro 639
SYSSTATE macro 641

T

TCBTOKEN macro 645
TESTART macro 651
time interval
  testing 577
TIME macro 655
timer
  setting a multiple 577
TIMEUSED macro 665
TOD (time-of-day) clock
  checking for synchronization with ETR 567
  converting value 561
  obtaining contents 567, 655
TRANMSG macro 669
TTIMER macro 683

U

UCB (unit control block)
  scanning 727
UCBDEVN macro 687
UCBINFO macro 691
UCBSCAN macro 727
UPDTMPB macro 739
user parameter
  passing 4

V

virtual storage
  bringing in a load module 457
  bringing in a program object 457
  loading 469, 481
  page-ahead function 469
  paging out 473, 481
  planning for future needs 469
  releasing contents 477, 481
Virtual storage
  sharing with IARVSERV macro 29
VRA (variable recording area)
  updating data 745
VRADATA macro 745

W

WAIT macro 751
WTL macro 757
WTO macro 765
WTOR macro 779

X

X-macros
  using 10
XCTL and XCTLX macros 791

Readers' Comments — We'd Like to Hear from You

z/OS
MVS Programming: Assembler Services
Reference, Volume 2 (IARR2V-XCTLX)

Publication No. SA22-7607-03

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
SA22-7607-03

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

SA22-7607-03

Cut or Fold
Along Line

Program Number: 5694-A01, 5655-G52

Printed in U.S.A.

SA22-7607-03