OS/390

MVS Programming: Authorized Assembler Services Reference, Volume 3 (LLACOPY-SDUMPX)

GC28-1766-08

OS/390

MVS Programming: Authorized Assembler Services Reference, Volume 3 (LLACOPY-SDUMPX)

GC28-1766-08

Note

Before using this information and the product it supports, be sure to read the general information under Appendix A, “Notices” on page 307.

Ninth Edition, December 2000

This is a major revision of GC28-1766-07.

This edition applies to Version 2 Release 10 of OS/390 (5647-A01) and to all subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

IBM welcomes your comments. A form for readers' comments may be provided at the back of this publication, 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/s390/os390/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 book
- 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, 2000. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About This Book vii
Who Should Use this Book vii
How to Use This Book vii
Where to Find More Information vii

Summary of Changes ix

Using the Services. 1

LLACOPY — Library Lookaside Refresh 25

LOAD — Bring a Load Module into Virtual Storage 31

LOADWAIT — Build a Wait State Parameter List for Use with WTO 39

LOCASCB — Locate Address Space Control Block (ASCB) Address 45

LXFRE — Free a Linkage Index 49

LXRES — Reserve a Linkage Index 53

MCSOPER — Manage Extended MCS Operations 57

MCSOPMSG — Retrieve MCS Operator Messages 73

MGCR — Issue an Internal START or REPLY Command 81

MGCRE — Issue Internal Commands 85

MIHQUERY — Retrieve MIH Time Interval 91

MODESET — Change System Status 97

NIL — Provide a Lock Via an AND IMMEDIATE (NI) Instruction 105

NMLDEF — Customizing the Nucleus 109

NUCLKUP — Nucleus Map Lookup Service 113

OIL — Provide a Lock Via an OR IMMEDIATE (OI) Instruction 117

OUTADD — Create an Output Descriptor 121

OUTDEL — Delete an Output Descriptor 135

PCLINK — Stack, Unstack, or Extract Program Call Linkage Information 147

PGANY — Page Anywhere 157

PGFIX — Fix Virtual Storage Contents 159

PGFIXA — Fix Virtual Storage Contents 163

PGFREE — Free Virtual Storage Contents 165

PGFREEA — Free Virtual Storage Contents 169

© Copyright IBM Corp. 1988, 2000

PGSER — Page Services 171

PGSER — Fast Path Page Services 179

POST — Signal Event Completion 183

PTRACE — Processor Trace 191

PURGEDQ — Purge SRB Activity 195

QEDIT — Command Input Buffer Manipulation 201

RACF Macros 203

RESERVE — Reserve a Device (Shared DASD) 205

RESMGR — Add or Delete a Resource Manager 217

RESUME — Resume Execution of a Suspended RB 227

RESUME — Resume or Purge a Suspended SRB 231

RISGNL — Issue Remote Immediate Signal 235

SCHEDIRB — Schedule IRB 239

SCHEDULE — Schedule a Service Request Block (SRB) 245

SCHEDXIT — Schedule an Exit Routine for Execution 251

SDUMP — Dump Virtual Storage 253

SDUMPX — Dump Virtual Storage 277

Appendix A. Notices 307

Index 309

Figures

1. Passing User Parameters in AR Mode . . . 4
2. Sample User Parameter List for Callers in AR Mode . . . 4
3. Issuing the Correct Macro Version at Execution Time . . . 10
4. Sample Macro Syntax Diagram . . . 14
5. Continuation Coding . . . 16
6. Sample Callable Service Syntax Diagram . . . 16
7. Service Summary . . . 18
8. Return and Reason Codes for the LLACOPY Macro . . . 27
9. Return Codes for the LOCASCB Macro . . . 46
10. Return Codes for the LXFRE Macro . . . 51
11. Return Code for the LXRES Macro . . . 55
12. Parameters available with REQUEST= services . . . 59
13. Parameters available with MSGDLVRY= services . . . 59
14. Return and Reason Codes for the MCSOPER Macro . . . 62
15. Parameters valid with REQUEST= services . . . 75
16. Return and Reason Codes for the MCSOPMSG Macro . . . 76
17. Parameters valid with REQUEST= services for the execute form of the macro . . . 79
18. Parameters valid with REQUEST= services for the modify form of the macro . . . 80
19. Return Codes for the START Command . . . 83
20. Return Codes for the START Command . . . 88
21. Return and Reason Codes for the MIHQUERY Macro . . . 93
22. Return and Reason Codes for the MODESET Macro . . . 102
23. Return Codes for the NUCLKUP Macro . . . 115
24. Return and Reason Codes for the OUTADD Macro . . . 125
25. Reason Codes for Return Code 04 . . . 125
26. Reason Codes for Return Code 08 . . . 126
27. Reason Codes for Return Code 0C . . . 130
28. Reason Codes for Return Code 10 . . . 134
29. Return and Reason Codes for the OUTDEL Macro . . . 137
30. Reason Codes for Return Code 04 . . . 138
31. Reason Codes for Return Code 08 . . . 139
32. Reason Codes for Return Code 0C . . . 142
33. Reason Codes for Return Code 10 . . . 146
34. Return Codes for the PGANY Macro . . . 158
35. Return Codes for the PGFIX Macro . . . 160
36. Return Codes for the PGFREE Macro . . . 166
37. Return Codes for the POST Macro . . . 188
38. Return Code for the PTRACE Macro . . . 193
39. Return Codes for the QEDIT Macro . . . 202
40. Return Code Area Used by RESERVE . . . 209
41. Return Codes for the RESERVE Macro with the RET=TEST Parameter . . . 210
42. Return Codes for the RESERVE Macro with the RET=USE Parameter . . . 210
43. Return Codes for the RESERVE Macro with the RET=HAVE Parameter . . . 211
44. Return Codes for the RESERVE Macro with the ECB Parameter . . . 211
45. Return Codes from the ADD Function . . . 221
46. Return Codes from the DELETE Function . . . 222
47. Return Codes for the RESUME Macro for RBs . . . 229
48. Return Codes for the RESUME Macro for SRBs . . . 233
49. Return Codes for the RISGNL Macro . . . 237
50. Return Codes for the SCHEDIRB Macro . . . 242
51. PSWREGS Parameter List . . . 259
52. Return Codes for the SDUMP Macro when BRANCH=NO . . . 267
53. Return Codes for the SDUMP Macro when BRANCH=YES . . . 267
54. Return Codes for the ECB Parameter and SRB Parameter . . . 267
55. Return Codes for the ECB or SRB Parameter with the DCB Parameter . . . 268
56. Reason Codes for Return Code 08 . . . 268

© Copyright IBM Corp. 1988, 2000

57. PSWREGS Parameter List 287
58. Return Codes for the SDUMPX Macro when BRANCH=NO 298
59. Return Codes for the SDUMPX Macro when BRANCH=YES 298
60. Return Codes for the ECB Parameter and SRB Parameter 298
61. Return Codes for the ECB or SRB Parameter with the DCB Parameter 299
62. Reason Codes for Return Code 08 299

About This Book

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

- Running in supervisor state
- Running under PSW key 0-7
- Residing in an APF-authorized library and link-edited with authorization code AC=1.

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

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

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

Who Should Use this Book

This book is for the programmer who is using assembler language to code a system program. A system program is usually one that runs in supervisor state or runs with PSW key 0-7 or resides on an APF-authorized library.

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

Assembler language programming is described in the following books:

- *HLASM Programmer's Guide*
- *HLASM Language Reference*

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

How to Use This Book

This book is one of the set of programming books 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 books, see *OS/390 Information Roadmap*.

Where to Find More Information

Where necessary, this book references information in other books, using shortened versions of the book title. For complete titles and order numbers of the books for all products that are part of OS/390, see *OS/390 Information Roadmap* (GC28-1727). The following table lists titles and order numbers for books related to other products.

© Copyright IBM Corp. 1988, 2000

Short Title Used in This Book	Title	Order Number
The Considerations of Physical Security in a Computer Environment	*The Considerations of Physical Security in a Computer Environment*	G520-2700

Summary of Changes

Summary of Changes
for GC-1766-08
as updated December, 2000.

This book contains information previously presented in GC28-1766-07, which also supports OS/390 Version 2 Release 10.

This revision reflects the deletion, addition, or modification of information to support miscellaneous maintenance items.

Summary of Changes
for GC28-1766-07
OS/390 Version 2 Release 10

This book contains information previously presented in GC28-1766-06, which supports OS/390 Version 2 Release 9.

This book includes 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.

Summary of Changes
for GC28-1766-06
OS/390 Version 2 Release 9

This book contains information previously presented in GC28-1766-05, which supports OS/390 Version 2 Release 7.

Support for the following APAR is included:

- OW41767

This book includes 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.

Summary of Changes
for GC28-1766-05
OS/390 Version 2 Release 7

This book contains information previously presented in GC28-1766-04, which supports OS/390 Version 2 Release 6.

Support for the following APAR is included:

- OW30729
- OW36902

This book includes 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.

Summary of Changes
for GC28-1766-04
OS/390 Version 2 Release 6

This book contains information previously presented in *OS/390 MVS Programming: Authorized Assembler Services Reference LLA-SDU*, GC28-1766-03, which supports OS/390 Version 2 Release 4.

This book includes 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.

© Copyright IBM Corp. 1988, 2000

Summary of Changes
for GC28-1766-03
OS/390 Version 2 Release 4

This book contains information previously presented in *OS/390 MVS Programming: Authorized Assembler Services Reference LLA-SDU*, GC28-1766-02, which supports OS/390 Version 1 Release 3.

This book includes 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.

Summary of Changes
for GC28-1766-02
OS/390 Version 1 Release 3

This book contains information previously presented in *OS/390 MVS Programming: Authorized Assembler Services Reference LLA-SDU*, GC28-1766-01, which supports OS/390 Release 2.

This book includes 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.

Summary of Changes
for GC28-1766-01
OS/390 Release 2

This book contains information previously presented in *OS/390 MVS Programming: Authorized Assembler Services Reference LLA-SDU*, GC28-1766-00, which supports OS/390 Release 1.

This book includes 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.

Summary of Changes
for GC28-1766-00
OS/390 Release 1

This book contains information previously presented in *MVS/ESA Programming: Authorized Assembler Services Reference, Volume 3 (LLACOPY-SDUMPX)*, GC28-1477, which supports MVS/ESA System Product Version 5.

Using the Services.

Macros and callable services are programming interfaces that application programs may 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
"Addressing Mode (AMODE)"	Both
"Address Space Control (ASC) Mode" on page 2	Both
"ALET Qualification" on page 3	Both
"User Parameters" on page 3	Macros
"Register Use" on page 4	Both
"Handling Return Codes and Reason Codes" on page 5	Both
"Handling Program Errors" on page 5	Both
"Handling Environmental and System Errors" on page 6	Both
"Selecting the Macro Level" on page 7	Macros
"Using X-Macros" on page 11	Macros
"Macro Forms" on page 12	Macros
"Coding the Macros" on page 13	Macros
"Coding the Callable Services" on page 16	Callable Services
"Including Equate (EQU) Statements" on page 16	Callable Services
"Link-Editing Linkage-Assist Routines" on page 17	Callable Services
"Service Summary" on page 17	Both

Addressing Mode (AMODE)

A program can execute in 24-bit addressing mode or 31-bit addressing mode. Regardless of the addressing mode that a program executes in, it can invoke most of the services described in this book.

In general, a program executing in 24-bit addressing mode cannot pass parameter addresses that are higher than 16 megabytes. However, there are exceptions. For example, a program executing 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.

In general, if a program running in 31-bit addressing mode issues a service, parameter addresses can be above or below 16 megabytes unless otherwise stated in the individual service description.

All callable services 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.

© Copyright IBM Corp. 1988, 2000

A program running in 31-bit addressing mode must use the MVS/SP Version 2 or later of the following macros:

ATTACH	MODESET
CALLDISP	SETRP
ESTAE	SYNCH
EVENTS	WTOR
FESTAE	

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 *OS/390 MVS Programming: Extended Addressability Guide* for more detailed information about AR mode.

Because the processor must resolve addresses differently for AR mode programs, **IBM recommends** that:

- **All** programs issue the SYSSTATE macro before issuing any other macros. Programs in primary mode must issue SYSSTATE ASCENV=P. Programs in AR mode must issue SYSSTATE ASCENV=AR.
- If your program switches from one ASC mode to another, your program should issue SYSSTATE immediately after the mode switch to indicate the new ASC mode.

Through the SYSSTATE macro, a program sets the SYSSTATE global variable (&SYSASCE) to indicate whether the program runs in primary or AR mode. Once a program has issued SYSSTATE, there is no need to reissue it unless the program switches ASC mode.

Some macros can generate code that is appropriate for programs in either primary mode or AR mode. These macros check &SYSASCE to determine which type of code to generate. If your program does **not** set &SYSASCE, any macros that check the variable set it to a default of primary mode upon assembly. Figure 7 on page 18 lists the macros that check &SYSASCE.

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.

Notes:

1. 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.
2. You can issue the SYSSTATE macro within your own user-written macro to determine whether your macro should generate code appropriate for primary or AR mode. See the SYSSTATE macro description in *OS/390 MVS Programming: Assembler Services Reference* for further details.
3. Callable services do not check &SYSASCE. To determine whether a callable service supports callers in AR or primary mode, check the individual service description.

ALET Qualification

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

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

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)
- An ALET for a private entry on the PASN-AL or the DU-AL.

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

User Parameters

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

Notes:

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

The macros shown in Figure 1 allow a caller in AR mode to pass information in the form of a parameter list (or parameter lists) to another routine. Figure 1 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.

Figure 1. Passing User Parameters in AR Mode		
Macro	**Parameter**	**Location of User Parameter List Address**
ATTACH/ATTACHX	PARAM,VL=1	AR/GPR 1 contains the address of a list of addresses and ALETs. (See Figure 2 for the format of the list.)
ESTAEX	PARAM	SDWAPARM contains the address of an 8-byte area, which contains the address and ALET of the parameter list.

When a caller in AR mode passes ALET-qualified addresses to the called program through PARAM,VL=1 on the ATTACH/ATTACHX macro, the system builds a list formatted as shown in Figure 2. 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 2 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
```

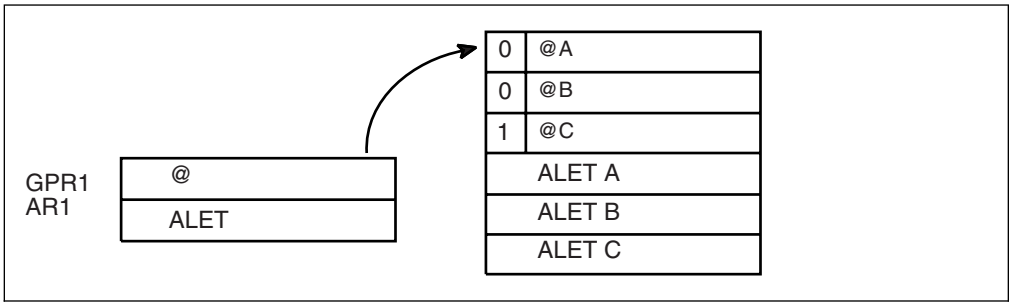


Figure 2. Sample User Parameter List for Callers in AR Mode

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.

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 *OS/390 MVS Programming: Authorized Assembler Services Guide* for information about writing recovery routines.

Handling Environmental and System Errors

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

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

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

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

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

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

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

Selecting the Macro Level

When MVS introduces a new version or a new release of an existing version, the new version or release usually supports all MVS macros from previous versions and releases. In general, programs assembled on an earlier version and release of MVS that issue macros will run on later versions and releases 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. When this is not true, it is called a **downward incompatibility**.

Placed in your program, the SPLEVEL macro helps you to deal with these downward incompatibilities. See “Service Summary” on page 17 for a description of the ways in which individual macros interact with the SPLEVEL macro. There are three categories of information provided in the summary table:

none The macro has no downward incompatibilities. SPLEVEL is not examined.

n A number in the range 2 to 4, indicating the release in which a downward incompatibility was introduced:

- **2** The macro is downward-incompatible between MVS/XA SP2.1.0 and prior releases
- **3** The macro is downward-incompatible between MVS/ESA SP3.1.0 and prior releases
- **4** The macro is downward-incompatible between MVS/ESA SP4.1.0 and prior releases
- **5** The macro is downward-incompatible between MVS/ESA SP5.1.0 and prior releases
- **6** The macro is downward-incompatible between OS/390 Release 2 and prior releases

AR The macro is downward incompatible between MVS/ESA SP3.1.0 and prior releases when the program specifies SYSSTATE ASCENV=AR.

Note: The problem of downward 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 10.

Some MVS macros are **downward incompatible** between MVS/XA SP2.1.0 and prior releases of MVS. When you assemble a program that issues these macros on an MVS/XA SP2.1.0 or higher system and attempt to run the program on version 1, these macros produce downward-incompatible statements, even though your program requests only function supported by version 1. Consequently, the program might not run as expected. Also, the version 1 expansions of these macros do not run on an MVS/XA SP2.1.0 or higher system if the caller is in 31-bit addressing mode. So, programs in 31-bit addressing mode must always use the MVS/XA SP2.1.0 or higher expansion of these macros.

The authorized macros that are downward incompatible between version 1 and MVS/XA SP2.1.0 are:

- ATTACH
- ESTAE
- EVENTS
- FESTAE
- SCHEDULE SCOPE=GLOBAL
- SDUMP
- SETFRR

- SETLOCK
- SJFREQ
- WTOR

In addition,

- macro CALLDISP is downward incompatible between MVS/ESA SP3.1.0 and prior releases; and
- the SETLOCK macro is downward incompatible between MVS/ESA SP4.1.0 and prior releases when TYPE=HIER is requested.

Many macros are sensitive to the difference between MVS/ESA SP3.1.0 and prior releases when SYSSTATE ASCENV=AR. **Set SYSSTATE ASCENV=AR when your program runs in AR ASC mode, and not otherwise.**

If you use installation- or vendor-written macros, some of these macros might have incompatibilities between versions and releases as well. Check your installation or vendor documentation to determine if such incompatibilities exist.

To manage the problem of incompatibilities between versions and releases of macros, use the SPLEVEL macro. The SPLEVEL macro with the SET=*n* parameter allows your program to select which level of a macro the assembler will generate.

Before issuing a downward-incompatible macro (for example, WTOR) , a program can specify the macro level by issuing SPLEVEL SET=*n*. For example, you could assemble the program using the OS/390 macro library, but indicate that you want to make sure the WTOR expansion will work on a release prior to MVS/XA SP2.1.0 by issuing SPLEVEL SET=1.

You can also use SPLEVEL with the TEST parameter when you write your own macros to ensure that the macro level for your macro is set. See the SPLEVEL macro description in *OS/390 MVS Programming: Assembler Services Reference* for more information.

According to the indication in the service summary table for a particular macro, use SPLEVEL as follows:

Indication	SPLEVEL
none	Specify SPLEVEL SET
2	If you need the expansion to work on a release prior to MVS/XA SP2.1.0, specify SPLEVEL SET=1
3	If you need the expansion to work on a release prior to MVS/ESA SP3.1.0, specify SPLEVEL SET=2
4	If you need the expansion to work on a release prior to MVS/ESA SP4.1.0, specify SPLEVEL SET=3
5	If you need the expansion to work on a release prior to MVS/ESA SP5.1.0, specify SPLEVEL SET=4
6	If you need the expansion to work on a release prior to OS/390 Release 2, specify SPLEVEL SET=5
AR	If you need the expansion to work on a release prior to MVS/ESA SP3.1.0, specify SPLEVEL SET=1 or SPLEVEL SET=2 and specify (or default to) SYSSTATE ASCENV=P

A program must select the level of the macro at execution time, based on the level of the operating system on which the program runs. The example in Figure 3 on page 10 shows one method of doing this. The example uses the WTOR macro but would work for any downward-incompatible macro. The WTOR macro downward incompatibility was introduced in MVS/XA SP2.1.0, and the macro has had new **function** added in each of MVS/XA SP2.1.0, MVS/ESA SP3.1.0, MVS/ESA SP4.1.0, and MVS/ESA SP5.1.0. So in this example, you could actually code the WTOR macro with different **parameters** depending on the level of MVS on which the program runs. This example assumes that you assemble the program on an OS/390 system. If you were coding WTOR with only version 1 parameters, you could

simply use SPLEVEL SET=1 for programs that run on version 1, and SPLEVEL SET (take the default) for programs that run on any other version of MVS.

The program in this example tests various bits in the CVT data area:

1. The program first tests the CVTOSEXT bit in CVTDCB to see if the CVT extension, which contains the bits that indicate whether the operating system is version 3 or higher, is present.
   a. If the extension is present, the program then determines if the operating system supports the MVS/ESA SP5.1.0 functions by testing the CVTH5510 bit in the CVTOSLV1 field. The CVTH5510 bit is 1 when the operating system is MVS/ESA SP5.1.0 or later. In the path where the CVTH5510 bit is 1, the WTOR macro can specify functions made available in MVS/ESA SP5.1.0 or a prior release.
   b. If the extension is present, but the MVS/ESA SP5.1.0 indicator is 0, then the program determines if the operating system supports the MVS/ESA SP4.1.0 functions by testing the CVTH4410 bit in CVTOSLV0. The CVTH4410 bit is 1 when the operating system is MVS/ESA SP4.1.0 or later. In the path where the CVTH4410 bit is 1, the WTOR macro can specify functions made available in MVS/ESA SP4.1.0 or a prior release.
   c. If the extension is present, but the version is neither 5 nor 4, then the program determines if the operating system supports the MVS/ESA SP3.1.0 functions by testing the CVTH3310 bit in CVTOSLV0. The CVTH3310 bit is 1 when the operating system is MVS/ESA SP3.1.0 or later. In the path where the CVTH3310 bit is 1, the WTOR macro can specify functions made available in MVS/ESA SP3.1.0 or a prior release.
2. If the extension is not present, the program tests the CVTMVSE bit in CVTDCB to determine if the operating system is version 2 or version 1. If the CVTMVSE bit is 1, the operating system is version 2; if the CVTMVSE bit is 0, the operating system is version 1. In the path where the CVTMVSE bit is 1, the WTOR macro can specify functions made available in MVS/XA SP2.1.0 or a prior release.

   In the path where the CVTMVSE bit is 0, it is important to specify SYSSTATE SET=1, because the downward incompatibility lies between MVS/XA SP2.1.0 and earlier releases. The WTOR macro can specify only functions made available prior to MVS/XA SP2.1.0.

```
         SPLEVEL SET              Set to current SPLEVEL
*
* Determine which system is executing.
         TM    CVTDCB,CVTOSEXT    Is extension present?
         BNO   SP2CHK             No, check for version 2.
         TM    CVTOSLV1,CVTH5510  Is the operating system version 5?
         BO    SP5                Yes, use version 5 macro.
         TM    CVTOSLV0,CVTH4410  Is the operating system version 4?
         BO    SP4                Yes, use version 4 macro.
         TM    CVTOSLV0,CVTH3310  Is the operating system version 3?
         BO    SP3                Yes, use version 3 macro.
*
SP2CHK   EQU   *
         TM    CVTDCB,CVTMVSE     Is the operating system version 2?
         BO    SP2                Yes, use version 2 macro.
         B     SP1                The operating system is version 1.
*
* Issue the version 5 expansion of the macro.
SP5      EQU   *
         WTOR  version-5-function(s)
         B     CONTINUE
*
* Issue the version 4 expansion of the macro.
SP4      EQU   *
         WTOR  version-4-functions(s)  Note: use only parameters that  X
                                       are supported in version 4.
         B     CONTINUE
*
* Issue the version 3 expansion of the macro.
SP3      EQU   *
         WTOR  version-3-functions(s)  Note: use only parameters that  X
                                       are supported in version 3.
         B     CONTINUE
*
* Issue the version 2 expansion of the macro.
SP2      EQU   *
         WTOR  version-2-functions(s)  Note: use only parameters that  X
                                       are supported in version 2.
         B     CONTINUE
*
* Issue the version 1 expansion of the macro.
SP1      EQU   *
         SPLEVEL SET=1
         WTOR  version-1-functions(s)  Note: use only parameters that  X
                                       are supported in version 1.
*
* Reset SPLEVEL for macro expansion to default.
*
CONTINUE EQU   *
         SPLEVEL SET
```

Figure 3. Issuing the Correct Macro Version at Execution Time

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. See “Selecting the Macro Level” on page 7 for more information on selecting a macro level.

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 installation. 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, 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' parameters, the macro will not assemble.
- If you code plistver='*n*' and do not specify any version '*n*' parameters, 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.

Using X-Macros

Some MVS services support callers in both primary and AR ASC mode. When the caller is in AR mode, macros must generate larger parameter lists; the increased size of the list reflects the addition of ALETs to qualify addresses, as described under "ALET Qualification" on page 3. For some MVS macros, two versions of a particular macro are available: one for callers in primary mode and one for callers in AR mode. The name of the macro for the AR mode caller is the same as the name of the macro for primary mode callers, except the AR

mode macro name ends with an “X”. This book refers to these macros as **X-macros**. The authorized X-macros are:

- ATTACHX
- ESTAEX
- SDUMPX
- SYNCHX

The only way these macros know that a caller is in AR mode is by checking the global symbol that the SYSSTATE macro sets. Each of these macros (and corresponding non-X-macro) checks the symbol. If SYSSTATE ASCENV=AR has been issued, the macro issues code that is valid for callers in AR mode. If it has not been issued, the macro generates code that is not valid for callers in AR mode. When your program returns to primary mode, use the SYSSTATE ASCENV=P macro to reset the global symbol.

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

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

- Callers in primary mode can invoke either macro.

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

- Callers in AR mode should issue the X-macros.

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

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

Macro Forms

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

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

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

Through the use of the different macro forms, a program that runs in a multiprogramming environment can avoid errors related to reenterable programs. The techniques required for using the macro forms, however, are different for some macros, called alternative list form macros, than for most other macros. For the alternative list form macros, the list form description notes that different techniques are required and refers you to the information under “Alternative List Form Macros” on page 13.

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 4 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.

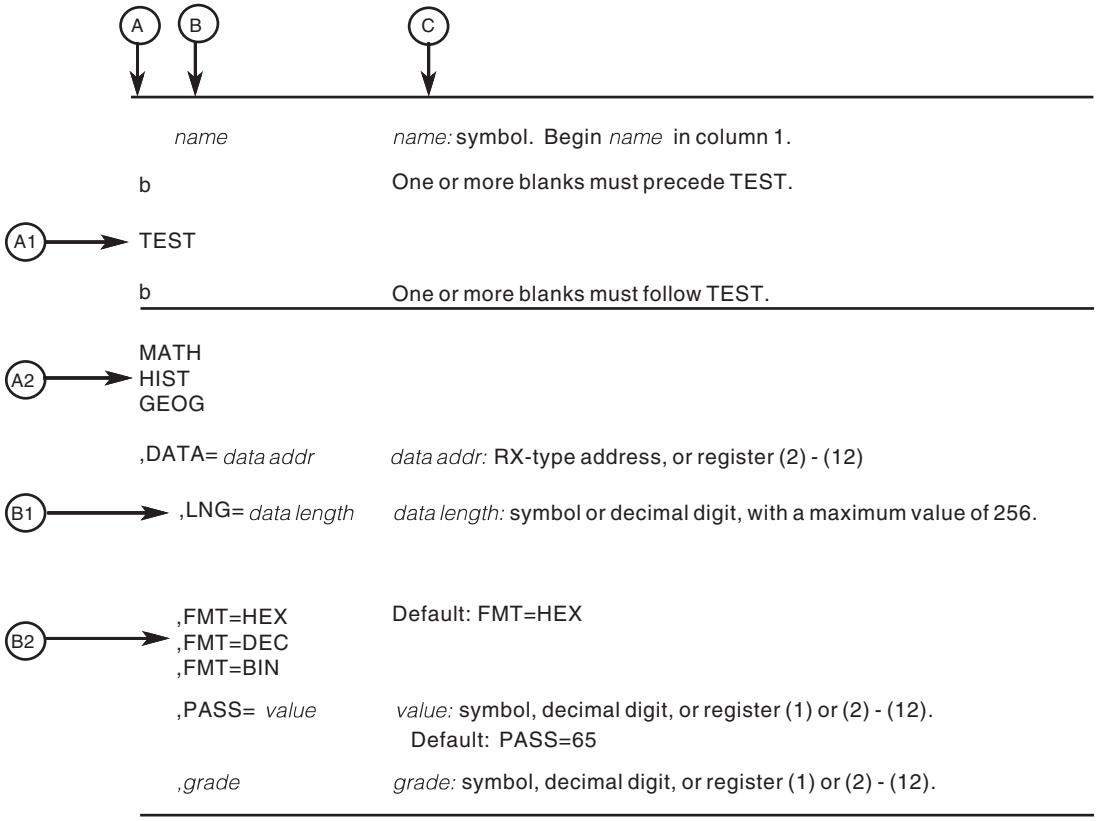


Figure 4. Sample Macro Syntax Diagram

- The first column, A, contains those parameters that are required for that macro. If a single line appears in that column, A1, the parameter on that line is required and you must code it. If two or more lines appear together, A2, the parameters on those lines are mutually exclusive, so you must code only one of those parameters.
- The second column, B, contains those parameters that are optional for that macro. If a single line appears in that column, B1, the parameter on that line is optional. If two or more lines appear together, B2, the parameters on those lines are mutually exclusive, so if you elect to make an entry, you must code only one of those parameters.
- 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 5 on page 16 shows an example of each method.

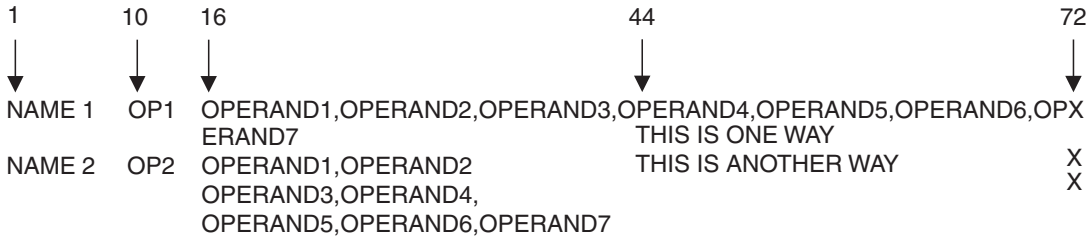


Figure 5. Continuation Coding

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)`. Figure 6 shows the syntax diagram for the sample callable service SCORE.

Figure 6. Sample Callable Service Syntax Diagram

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

Considerations for coding callable services are:

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

Including Equate (EQU) Statements

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

Note: Check the “Programming Requirements” section of the individual service description to determine if the equate statements are available for the callable service you are using. If the equate statements are available, that section will also provide a list of the statements that are provided, along with a description of how to include them in your program.

Link-Editing Linkage-Assist Routines

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

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

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

Note: 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

Figure 7 on page 18 lists the services described in the following:

- *OS/390 MVS Programming: Authorized Assembler Services Reference ALE-DYN*
- *OS/390 MVS Programming: Authorized Assembler Services Reference ENF-IXG*
- *OS/390 MVS Programming: Authorized Assembler Services Reference LLA-SDU*
- *OS/390 MVS Programming: Authorized Assembler Services Reference SET-WTO*.

For each service, the table indicates:

- Whether a program in AR ASC mode can issue the service
- Whether a program in cross memory mode can issue the service
- Whether the macro checks the SYSSTATE global variable
- Whether the macro checks the SPLEVEL global variable.

Notes:

1. 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 *OS/390 MVS Programming: Extended Addressability Guide*.
2. A program running in primary ASC mode when PASN=SASN=HASN can issue any of the services listed in the table.

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

Figure 7 (Page 1 of 7). Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
ALESERV	Yes	Yes	No	No
ASCRE	Yes	Yes	Yes	No
ASDES	Yes	Yes	Yes	No
ASEXT	Yes	Yes	No	No
ATSET	No	Yes	No	No
ATTACH	Yes (See note 1 on page 24)	No	Yes	Yes
ATTACHX	Yes	No	Yes	No
AXEXT	No	Yes	No	No
AXFRE	No	Yes	No	No
AXRES	No	Yes	No	No
AXSET	No	Yes	No	No
BPXEKDA	Yes	No	Yes	No
BPXESMF	Yes	No	Yes	Yes
CALLDISP	No	Yes	No	Yes
CALLRTM	No	Yes (See note 2 on page 24)	No	No
CHANGKEY	No	Yes	No	No
CIRB	No	No	No	No
CMDAUTH	No	No	No	No
COFCREAT	Yes	Yes	Yes	Yes
COFDEFIN	Yes	Yes	Yes	Yes
COFIDENT	Yes	Yes	Yes	Yes
COFNOTIF	Yes	Yes	Yes	Yes
COFPURGE	Yes	Yes	Yes	Yes
COFREMOV	Yes	Yes	Yes	Yes
COFRETRI	Yes	Yes	Yes	Yes
COFSDONO	No	No	Yes	Yes
CONFCHG	No	No	Yes	Yes
CPF	No	No	No	No
CPOOL	No	Yes	No	No
CSRSI	No	Yes	No	No
CSRUNIC	Yes	Yes	No	No
CSVAPF	Yes (See note 11 on page 24)	Yes (See note 12 on page 24)	Yes	No
CSVDYNEX	Yes (See note 13 on page 24)	Yes (See note 14 on page 24)	Yes	No
CTRACE	No	No	Yes	Yes
CTRACECS	Yes	No	Yes	Yes
CTRACEWR	Yes	Yes	Yes	Yes
DATOFF	Yes	No	No	No
DEQ	No	No	No	No

Figure 7 (Page 2 of 7). Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
DIV	Yes	No	Yes	No
DOM	No	No	No	No
DSPSERV	Yes	Yes	Yes	Yes
DYNALLOC	No	No	No	No
ENFREQ	No	No	No	No
ENQ	No	No	No	No
ESPIE	No	No	No	No
ESTAE (See note 3 on page 24)	No	No	Yes	Yes
ESTAEX	Yes	Yes	Yes	Yes
ETCON	No	Yes	No	No
ETCRE	No	Yes	No	No
ETDEF	Yes	Yes	No	No
ETDES	No	Yes	No	No
ETDIS	No	Yes	No	No
EVENTS	No	No	No	Yes
EXTRACT	No	No	No	No
FESTAE	No	No	No	Yes
FREEMAIN	Yes (See note 4 on page 24)	Yes	Yes	No
GETDSAB	No	No	Yes	Yes
GETMAIN	Yes (See note 4 on page 24)	Yes	Yes	No
GQSCAN	No	Yes	No	No
GTRACE	No	Yes	No	No
HSPSERV	Yes	Yes (See note 5 on page 24)	(See note 6 on page 24)	Yes
IARR2V	Yes	Yes	No	No
IARSUBSP	Yes	Yes	Yes	Yes
IARVSERV	Yes	Yes	Yes	Yes
IAZXJSAB	Yes	Yes (See note 15 on page 24)	Yes	No
IEAARR	Yes	Yes	Yes	No
IEALSQRY	Yes	Yes	Yes	No
IEAMRMF3	No	Yes	No	No
IEAMSCHD	Yes	Yes	Yes	Yes
IEANTCR	Yes	Yes	N/A	N/A
IEANTDL	Yes	Yes	N/A	N/A
IEANTRT	Yes	Yes	N/A	N/A
IEARBUP	Yes	Yes	Yes	Yes
IEATDUMP	Yes	No	Yes	Yes
IEAVAPE	No	Yes	No	No
IEAVDPE	No	Yes	No	No

Figure 7 (Page 3 of 7). Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
IEAVPSE	No	Yes	No	No
IEAVRLS	No	Yes	No	No
IEAVRPI	No	Yes	No	No
IEAVTPE	No	Yes	No	No
IEAVXFR	No	Yes	No	No
IEEQEMCS	Yes	Yes	Yes	Yes
IEEVARYD	No	No	Yes	Yes
IEFPPSCN	No	No	Yes	Yes
IEFQMREQ	No	No	No	No
IEFSSI	Yes	No	No	No
IEFSSVT	Yes	No	No	No
IEFSSVTI	Yes	Yes	No	No
IOSADMF	No	No	Yes	Yes
IOSCAPF	No	Yes (See note 7 on page 24)	Yes	Yes
IOSCAPU	Yes	Yes (See note 7 on page 24)	Yes	Yes
IOSCDR	No	No	Yes	Yes
IOSCHPD	Yes	Yes	Yes	Yes
IOSCMXA	No	Yes (See note 7 on page 24)	Yes	Yes
IOSCMXR	No	Yes (See note 7 on page 24)	Yes	Yes
IOSDCXR	No	Yes (See note 7 on page 24)	Yes	Yes
IOSENQ	Yes	Yes	Yes	Yes
IOSINFO	No	No	No	No
IOSLOOK	No	No	No	No
IOSPTHV	No	No	Yes	Yes
IOSUPFA	No	Yes	Yes	Yes
IOSUPFR	No	Yes	Yes	Yes
IOSWITCH	Yes	Yes	Yes	Yes
ISGLCRT	No	Yes	N/A	N/A
ISGLOBT	No	Yes	N/A	N/A
ISGLREL	No	Yes	N/A	N/A
ISGLPRG	No	Yes	N/A	N/A
ITTFMTB	No	No	No	No
ITZXFILT	No	Yes	Yes	No
IWMCLSFY	No	Yes	Yes	Yes
IWMCONN	No	Yes	Yes	Yes
IWMDISC	No	Yes	Yes	Yes
IWMECQRY	No	Yes	Yes	Yes
IWMECREA	No	Yes	Yes	Yes

Figure 7 (Page 4 of 7). Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
IWMEDELE	No	Yes	Yes	Yes
IWMMABNL	No	Yes	No	No
IWMMCHST	No	Yes	No	No
IWMMCREA	No	Yes	Yes	Yes
IWMMDELE	No	Yes	Yes	Yes
IWMMEXTR	No	Yes	Yes	Yes
IWMMINIT	No	Yes	No	No
IWMMNTFY	No	Yes	Yes	Yes
IWMMRELA	No	Yes	Yes	Yes
IWMMSWCH	No	Yes	Yes	Yes
IWMMXFER	No	Yes	No	No
IWMPQRY	Yes	Yes	Yes	Yes
IWMRCOLL	Yes	Yes	Yes	Yes
IWMRPT	No	Yes	Yes	Yes
IWMRQRY	Yes	Yes	Yes	Yes
IWMSRDRS	No	Yes	Yes	Yes
IWMSRSRG	No	Yes	Yes	Yes
IWMSRSRS	No	Yes	Yes	Yes
IWMWMCON	No	Yes	Yes	Yes
IWMWQRY	Yes	Yes	Yes	Yes
IWMWQWRK	No	Yes	Yes	Yes
IXCCREAT	Yes	Yes	Yes	Yes
IXCDELET	Yes	Yes	Yes	Yes
IXCJOIN	Yes	No	Yes	Yes
IXCLEAVE	Yes	No	Yes	Yes
IXCMG	Yes	Yes	Yes	Yes
IXCMOD	Yes	Yes	Yes	Yes
IXCMSGI	Yes	No	Yes	Yes
IXCMSGO	Yes	Yes	Yes	Yes
IXCQUERY	Yes	Yes	Yes	Yes
IXCQUIES	Yes	No	Yes	Yes
IXCSETUS	Yes	Yes	Yes	Yes
IXCTERM	Yes	Yes	Yes	Yes
LLACOPY	No	No	Yes	Yes
LOAD	Yes	No	No	No
LOADWAIT	No	Yes	Yes	No
LOCASCB	Yes	Yes	Yes	No
LXFRE	No	Yes	No	No
LXRES	No	Yes	No	No
MCSOPER	Yes	No	Yes	Yes
MCSOPMSG	Yes	No	Yes	Yes
MGCR	No	No	No	No

Figure 7 (Page 5 of 7). Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
MGCRE	No	No	No	No
MIHQUERY	Yes	No	Yes	No
MODESET	No	Yes	No	No
NIL	No	No	No	No
NMLDEF	No	No	No	No
NUCLKUP	No	No	No	No
OIL	No	No	No	No
OUTADD	No	No	No	No
OUTDEL	No	No	No	No
PCLINK	No	Yes	No	No
PGANY	No	No	No	No
PGFIX	No	Yes	No	No
PGFIXA	No	No	No	No
PGFREE	No	Yes	No	No
PGFREEA	No	No	No	No
PGSER	Yes (See note 8 on page 24)	Yes (See note 8 on page 24)	No	No
POST	No	Yes	No	No
PTRACE	No	Yes	No	No
PURGEDQ	No	No	No	No
QEDIT	No	No	No	No
RESERVE	No	No	No	No
RESMGR	Yes	Yes	No	No
RESUME	No	Yes	No	No
RISGNL	No	Yes	No	No
SCHEDIRB	Yes	No	Yes	No
SCHEDULE	Yes	Yes	Yes	Yes
SCHEDXIT	No	Yes	No	No
SDUMP	Yes (See note 1 on page 24)	Yes (See note 9 on page 24)	Yes	Yes
SDUMPX	Yes	Yes (See note 9 on page 24)	Yes	No
SETFRR	Yes	Yes	Yes	Yes
SETLOCK	Yes	Yes	Yes	Yes
SETRP	Yes	Yes	Yes	No
SJFREQ	No	Yes	No	Yes
SPIE	No	No	No	No
SPOST	No	No	No	No
SRBSTAT	No	Yes	No	No
SRBTIMER	No	No	No	No
STATUS	Yes	Yes	No	No
STORAGE	Yes	Yes	No	No
SUSPEND	No	Yes	No	No

Figure 7 (Page 6 of 7). Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL
SVCUPDTE	No	No	No	No
SWAREQ	No	No	No	No
SWBTUREQ	No	No	No	No
SYNCH	Yes (See note 1 on page 24)	No	Yes	No
SYNCHX	Yes	No	Yes	No
SYSEVENT	No	No	No	No
TCBTOKEN	Yes	Yes	No	No
TCTL	No	No	No	No
TESTAUTH	No	No	No	No
TIMEUSED	Yes (See note 10 on page 24)	Yes	No	No
T6EXIT	No	No	No	No
UCBINFO	Yes	Yes	Yes	No
UCBLOOK	Yes	Yes	Yes	Yes
UCBPIN	Yes	Yes	Yes	Yes
UCBSCAN	Yes	Yes	Yes	Yes
VSMLIST	No	Yes	No	No
VSMLOC	No	Yes	No	No
VSMREGN	No	Yes	No	No
WAIT	No	Yes	No	No
WTL	No	No	No	No
WTO	No	No	No	No
WTOR	No	No	No	Yes

Figure 7 (Page 7 of 7). Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Checks SPLEVEL

Notes:

1. Primary mode callers can use either macro in the following macro pairs:

   ATTACH or ATTACHX
   SDUMP or SDUMPX
   SYNCH or SYNCHX

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

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

LLACOPY — Library Lookaside Refresh

Description

The LLACOPY macro obtains new directory entries from DASD and uses them to synchronously refresh the LLA directory. LLACOPY uses the BLDL macro to obtain the directory entries, and returns them to the caller even if LLA is not active. LLACOPY requires the same input parameters as BLDL: an open DCB and a BLDL list of member names. If the directory entry for any of the member names is not found, that member will be removed from LLA's directory as part of the refresh.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state with any 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:	None

Programming Requirements

None.

Restrictions

The storage key of the parameter list and the storage key of the BLDL list must be the same as the PSW key in which the caller runs.

The caller must have UPDATE access to the data set in either the FACILITY class or the DATASET class. LLACOPY first checks to see if the caller is authorized in the FACILITY class. The resource name used in this check is in the form CSVLLA.data_set_name. If the caller is authorized (or if there is no profile protecting the resource name), LLACOPY completes successfully. If the caller is not authorized, LLACOPY then checks to see if the caller is authorized in the DATASET class. If the caller is authorized, LLACOPY completes successfully. Otherwise, LLACOPY fails, and an SMF record may be created by the external security product.

Input Register Information

Before issuing the LLACOPY 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	If GPR 15 contains a return code of X'8', GPR 0 contains a reason code; otherwise, used as a work register by the system
2-14	Unchanged
15	Return code

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

© Copyright IBM Corp. 1988, 2000

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

LLACOPY eliminates the reduced fetch I/O benefit of LLA's module caching until the module is again staged to LLA's VLF data space.

An additional cost of using LLACOPY for LLA-managed data sets is that LLA serializes the use of the LLA directory. So, for the duration of the LLACOPY, the LLA directory cannot be changed by another LLACOPY or LLA command.

Syntax

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

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

DCB=*dcb addr*	*dcb addr*: RX-type address or register (2) - (12).
,BLDLLIST=*list addr*	*list addr*: RX-type address or register (2) - (12).
,RETCODE=*ret code*	*ret code*: RX-type address or register (2) - (12).
,RSNCODE=*rsn code*	*rsn code*: RX-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

DCB=*dcb addr*
Specifies the address of an open DCB that LLACOPY uses to issue the BLDL macro to obtain new directory entries.

,BLDLLIST=*list addr*
Specifies the address of a list of member names in the format required by the BLDL macro.

,RETCODE=*ret code*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15.

,RSNCODE=*rsn code*
Specifies the location where the system is to store the reason code. If the return code is X'8', the reason code is also in GPR 0.

,MF=S

Specifies the standard form of LLACOPY. The standard form places the parameters into an in-line parameter list.

ABEND Codes

LLACOPY might abnormally terminate with abend code X'023'. See *OS/390 MVS System Codes* for an explanation of the reason codes and programmer responses for X'023'.

Return and Reason Codes

The return and reason codes for LLACOPY are the same as those for the BLDL macro. When control returns from LLACOPY, GPR 15 (and *ret code*, if you coded RETCODE) contains one of the following hexadecimal return codes. If you receive a return code of 8, GPR 0 (and *rsn code*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Figure 8. Return and Reason Codes for the LLACOPY Macro

Return Code	Reason Code	Meaning and Action
00	None	**Meaning**: LLACOPY found all requested directory entries and copied the new entries into the caller's BLDL list. If LLA was available, LLACOPY refreshed the LLA directory for the given members in the data set concatenation that the open DCB defined. **Action**: None.
04	None	**Meaning**: LLACOPY did not find all the requested directory entries, and might not have found any entries. It copies into the caller's BLDL list entries that it did find. If LLA was available, LLACOPY refreshed the LLA directory for the entries that it found, and removed from the LLA directory any members whose directory entries it did not find. **Action**: Ensure that each member name in the caller's BLDL list is in one of the data sets described by the caller's DCB.
08	00	**Meaning**: Environmental error. LLACOPY detected a permanent I/O error when trying to search the directory. LLACOPY did not update the BLDL list or refresh the LLA directory. **Action**: Contact your system programmer. The error could be caused by a software or hardware problem.
08	04	**Meaning**: Environmental error. LLACOPY did not have sufficient virtual storage in the primary address space to complete. LLACOPY did not update the BLDL list or refresh the LLA directory. **Action**: Contact your system programmer, who can ensure that sufficient virtual storage is available.

Example

Request LLACOPY to retrieve and update module ABC from library USERLIB. USERLIB is opened by the application program. The DCB that was used to OPEN the library is also used in the LLACOPY.

```
           LLACOPY BLDLLIST=B_LIST,DCB=USERDCB,
                   RETCODE=RETNCODE,RSNCODE=RSONCODE

USERDCB  DCB  DDNAME=USERLIB,MACRF=R,DSORG=PO
B_LIST   DS   0F                BLDL LIST
         DC   H'01'             NUMBER OF ENTRIES
         DC   H'76'             LENGTH OF ENTRY
MODNAME  DC   CL8'ABC     '     MODULE NAME
         DS   CL68              DIRECTORY INFO FILLED IN BY LLACOPY
RETNCODE DS   F
RSONCODE DS   F
```

LLACOPY—List Form

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

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

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 LLACOPY macro with the following exception:

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

list addr is the address of the storage area for the parameter list.

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

LLACOPY—Execute Form

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

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

DCB=*dcb addr*	*dcb addr*: RX-type address or register (2) - (12).

,BLDLLIST=*list addr*	*list addr*: RX-type address or register (2) - (12).
,RETCODE=*ret code*	*ret code*: RX-type address or register (2) - (12).
,RSNCODE=*rsn code*	*rsn code*: RX-type address 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 LLACOPY macro with the following exception:

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

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

LOAD — Bring a Load Module into Virtual Storage

Description

The LOAD macro brings 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 RMODE of the module. 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 described in *OS/390 MVS Programming: Assembler Services Reference*), *and* there is no other requirement for the module.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state or supervisor state, and any PSW key. The GLOBAL, EOM, ADDR, and ADRNAPF parameters are restricted to authorized users (APF authorized, PSW key 0-7, or supervisor state).
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

If you code any of the parameters LSEARCH, ADDR, ADRNAPF, GLOBAL, EOM, 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 data set directory or it must have been specified previously in an IDENTIFY macro invocation. If the LOAD macro cannot find the specified entry name, the caller's task is ended abnormally unless the caller provides an ERRET exit.
- 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 LOAD is successful, the GPRs contain the following when control returns to the caller:

© Copyright IBM Corp. 1988, 2000

Register	Contents
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. The other three bytes contain the module length in doublewords. When the module is a program object, bound with FETCHOPT=NOPACK option, multiplying by eight the returned length value (to get the total number of bytes) always results in a multiple of one page (4096 bytes) (this is the full size of the area obtained with GETMAIN to hold the program object). If the program object is bound with FETCHOPT=PACK, the length value returned is the virtual storage size indicated in the directory entry. See *OS/390 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.

Performance Implications

None.

Syntax

The LOAD macro is written as follows:

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

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name*: Symbol. *entry name addr*: RX-type address or register (0), (2) - (12); A-type address or register (2) - (12). *list entry addr*: RX-type address, or register (2) - (12); A-type address or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: RX-type address, or register (1) or (2) - (12); A-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**: LSEARCH=NO

,ADDR=*load addr* ,ADRNAPF=*load addr*	*load addr*: A-type address or register (2) - (12).
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default**: GLOBAL=NO If GLOBAL=YES is specified, the default is GLOBAL=(YES,P).
,EOM=NO ,EOM=YES	**Default**: EOM=NO **Note:** GLOBAL must be specified with EOM=YES.
,LOADPT=*addr*	*addr*: A-type address or register (2) - (12). **Note:** ADDR and ADRNAPF cannot be specified with LOADPT.
,RELATED=*value*	

Parameters

The parameters are explained below:

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.

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. The LOAD and the BLDL must be issued from the same task. Otherwise, the system might terminate the program with a system completion code of 106 and a return code of 15. Therefore, do not issue an ATTACH or DETACH between issuances of BLDL and LOAD.

,DCB=*dcb addr*

Specifies the address of the 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.

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 macros in the subtask 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. GPR 1 contains the abend code that would have resulted had the task abended, and GPR 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) you want the library search limited to the job pack area and to the first library in the normal search sequence.

,ADDR=*load addr*
,ADRNAPF=*load addr*

Specifies that the module is to be loaded beginning at the designated address. The address must begin on a doubleword boundary. Storage for the module must have been previously allocated in the key of the eventual user. The system does not search for the module and does not maintain a record of the module once it is loaded. If you code ADDR or ADRNAPF, you must also code the DCB parameter (not DCB=0) and you must not code GLOBAL or LOADPT.

Note: The RMODE of the load module must agree with this address. If the user specifies an address above 16 megabytes in virtual, the load module must have an RMODE of ANY.

If your program requires that the module be in an APF-authorized library, use ADDR; otherwise, use ADRNAPF.

- For the ADDR parameter, the system checks that the module being loaded is in an APF-authorized library.
- For the ADRNAPF parameter, the system does not check that the module resides in an APF-authorized library. Therefore, if the module is not in an APF-authorized library, the program must make sure that the loaded programs receive control only in problem state.

,GLOBAL=YES
,GLOBAL=(YES,P)
,GLOBAL=(YES,F)
,GLOBAL=NO

Specifies whether the module is to be loaded into the pageable common service area (CSA) (GLOBAL=(YES,P) or GLOBAL=YES), loaded into fixed CSA (GLOBAL=(YES,F)), or not loaded into CSA (GLOBAL=NO). (The module must not have been previously loaded into CSA with different attributes by the same job step, the module must also be reentrant and must reside in an APF-authorized library.) For GLOBAL=(YES,F), the module must not be marked as requiring alignment on a page boundary. If you code the GLOBAL parameter, you cannot code the ADDR or ADRNAPF parameter.

If the requested module resides in the link pack area, the LOAD request performs as though the GLOBAL parameter was omitted. The LOAD request locates the module in the link pack area, allows access to it, but does not load a copy of the desired module into the common service area.

Note: A load request with the GLOBAL=YES, (YES,P), or (YES,F) option does not cause the loaded module to be implicitly known to other address spaces. The loaded module can be accessed by other address spaces, however, only the task that loaded the module may delete it.

,EOM=YES
,EOM=NO

Indicates whether a module in global storage is to be deleted when the address space terminates (EOM=YES) or when the task terminates (EOM=NO). If you code EOM, you must also code GLOBAL.

,LOADPT=*addr*

Specifies that the starting address at which the module was loaded is to be returned to the caller at the indicated address. If you code LOADPT, you cannot code ADDR or ADRNAPF.

,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.

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 system completion 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 system completion code in GPR 1.

Example 1

Bring a load module with entry name PGMLKRUS into virtual storage. Let the system find the module from available libraries.

```
LOAD  EP=PGMLKRUS
```

Example 2

Bring a load module with entry name PGMEOM into pageable CSA storage and return the load address at location PGMLPT.

```
LDPGM    LOAD  EP=PGMEOM,EOM=YES,LOADPT=PGMLPT,GLOBAL=(YES,P)
         .
         .
         .
PGMLPT   DS    A                    LOAD ADDRESS RETURNED HERE
```

LOAD—List Form

The list form of the LOAD macro builds a nonexecutable parameter list that can be referred to 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.
␣	One or more blanks must precede LOAD.
LOAD	
␣	One or more blanks must follow LOAD.

EP=*entry name* EPLOC=*entry name addr* DE=*list entry addr*	*entry name*: Symbol. *entry name addr*: A-type address. *list entry addr*: A-type address.
,DCB=*dcb addr*	*dcb addr*: A-type address.
,LSEARCH=NO ,LSEARCH=YES	**Default**: LSEARCH=NO
,ADDR=*load addr* ,ADRNAPF=*load addr*	*load addr*: A-type address.
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default**: GLOBAL=NO If GLOBAL=YES is specified, the default GLOBAL=(YES,P).
,EOM=NO	**Default**: EOM=NO

,EOM=YES	**Note:** GLOBAL must be specified with EOM=YES.
,LOADPT=*addr*	*addr*: A-type address. **Note**: ADDR and ADRNAPF cannot be specified with LOADPT.
,RELATED=*value*	
,SF=L	

Parameters

The parameters are explained under the standard form of 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* 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).
,ERRET=*err rtn addr*	*err rtn addr*: RX-type address, or register (2) - (12).
,LSEARCH=NO ,LSEARCH=YES	**Default**: LSEARCH=NO
,ADDR=*load addr* ,ADRNAPF=*load addr*	*load addr*: RX-type address or register (2) - (12). **Note:** For an RX-type address, the operand is treated as the address of a field that contains the actual load address.
,GLOBAL=YES ,GLOBAL=(YES,P) ,GLOBAL=(YES,F) ,GLOBAL=NO	**Default**: GLOBAL=NO **Note:** If GLOBAL=YES is specified, the default is GLOBAL=(YES,P).
,EOM=NO ,EOM=YES	**Default**: EOM=NO **Note:** GLOBAL must also be specified with EOM=YES.
,LOADPT=*addr*	*addr*: RX-type address or register (2) - (12).

Note: ADDR and ADRNAPF cannot be specified with LOADPT.

,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 LOAD macro with the following exception:

,SF=(E,*list addr*)
Specifies the execute form of the LOAD macro.

LOADWAIT — Build a Wait State Parameter List for Use with WTO

Description

The LOADWAIT macro can:

- Define storage for a parameter list
- Define and initialize storage for a parameter list
- Modify storage of an existing parameter list.

OS/390 MVS Programming: Authorized Assembler Services Guide describes how to use the LOADWAIT macro.

The WSPARM parameter of the WTO macro contains the name of the parameter list that you build using the LOADWAIT macro. WTO uses the parameter list from LOADWAIT to put the system into the wait state and issues one message to the operator. The wait state code and operator message explain what action the operator is to take. For more information about wait state codes, see *OS/390 MVS System Codes*.

There is a list and modify form of the macro, but no standard form.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0, or APF-authorized
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	No requirement

Programming Requirements

None.

Restrictions

The LOADWAIT parameter list and action code receiving byte, if specified, must be in fixed storage of the WTO issuer's address space.

Register Information

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

When control returns to the caller after the caller has issued the modify form of the macro, the general purpose registers contain:

Register	Contents
0	Address of the action code variable if specified.
1	Address of parameter list.
2-15	Unchanged

© Copyright IBM Corp. 1988, 2000

Performance Implications

None.

LOADWAIT—List Form

Use the list form of the LOADWAIT macro together with the modify form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage or initializes that storage. The modify form of the macro updates the parameters in the area previously defined by the list form.

Syntax

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

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

	Valid parameters (Required parameters are underlined)
WAITTYPE=RESTARTABLE	CODE, REASON, PSAPARM, MF
WAITTYPE=NONREST	CODE, REASON, MF
,CODE=*wait state code*	*wait state code*: Constant
,REASON=*reason code*	*reason code*: Constant
,REASON=0	**Default**: REASON=0
,PSAPARM=*PSA parm*	*PSA parm*: Constant
,PSAPARM=0	**Default**: PSAPARM=0
,MF=(L,*list addr,attr*)	*list addr*: RX-type address. *attr*: 1- to 60-character input string. **Default**: 0F

Parameters

The parameters are explained as follows:

WAITTYPE=RESTARTABLE
WAITTYPE=NONREST
Identifies the type of wait state to be loaded. WAITTYPE=RESTARTABLE indicates a restartable wait state. WAITTYPE=NONREST indicates a nonrestartable wait state.

,CODE=*wait state code*
Specifies a 2-byte wait state code. The contents of the leftmost 4 bits are irrelevant; the remaining 12 bits contain the wait state code. For more information about wait state codes, see *OS/390 MVS System Codes*.

,REASON=*reason code*
,REASON=0
Specifies a 2-byte wait state reason code. For more information about wait states and their reason codes, see *OS/390 MVS System Codes*.

,PSAPARM=*PSA parm*
,PSAPARM=0
Specifies a fullword field that you can use for additional information, such as a pointer to a data area, or a system ID at the time you issue the LOADWAIT macro. This information is used for diagnostic purposes when the system enters a wait state. The PSAPARM parameter is valid only if you specify WAITTYPE=RESTARTABLE.

If you do not specify PSAPARM, the system initializes the field to zeroes.

,MF=(L,*list addr,attr***)**
Specifies the list form of the LOADWAIT macro. *list addr* names the area that the system is to use for the parameter list. Use standard assembler variable naming conventions to name this area, and refer to the parameter list by the same name. Use this area name as input on the WSPARM parameter of the WTO macro.

attr is an optional 1- to 60-character 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 0F, which forces the parameter list to a fullword boundary.

Return and Reason Codes

None.

Example 1

Generate a parameter list to load a restartable wait state.

```
         LOADWAIT WAITTYPE=RESTARTABLE,CODE=WAIT062,MF=(L,WAITPRM)
.
.
.
         DS   0D
WAIT062  EQU  X'62'                * WAIT STATE CODE IS KNOWN
.
.
.
```

Example 2

Generate a parameter list to load a restartable wait state and specify a reason code and PSAPARM.

```
         LOADWAIT WAITTYPE=RESTARTABLE,CODE=WAIT114,REASON=REASON02,
                   PSAPARM=OPERINFO,MF=(L,WAITPRM2)
.
.
.
         DS   0D
WAIT114  EQU  X'114'               * WAIT STATE CODE IS KNOWN
REASON02 EQU  X'2'                 * REASON CODE IS KNOWN
OPERINFO EQU  X'C5E2C1E3'          * PSAPARM = 'ESAT'
.
.
.
```

Example 3

Generate a parameter list to load a nonrestartable wait state.

```
         LOADWAIT WAITTYPE=NONREST,CODE=WAIT093,MF=(L,WAITPRM3)
.
.
.
         DS   0D
WAIT093  EQU  X'093'               * WAIT STATE CODE
.
.
.
```

LOADWAIT—Modify Form

The modify form of the macro updates an existing parameter list.

Note: When you use the modify form of the macro, the parameter list is reset to all zeroes. You must specify all of the required information each time you use the modify form.

Syntax

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

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

	Valid parameters (Required parameters are underlined)
WAITTYPE=RESTARTABLE	CODE, REASON, ACTCODE, PSAPARM, MF
WAITTYPE=NONREST	CODE, REASON, MF
,CODE=*wait state code*	*wait state code*: Value
,REASON=*reason code*	*reason code*: Value
,REASON=0	**Default**: REASON=0
,ACTCODE=*action code*	*action code*: 1-byte field
,ACTCODE=0	**Default**: ACTCODE=0
,PSAPARM=*PSA parmr*	*PSA parm*: Value
,PSAPARM=0	**Default**: PSAPARM=0
,MF=(M,*list addr*)	*list addr*: RX-type address.

Parameters

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

,ACTCODE=*action code*
,ACTCODE=0
Specifies a 1-byte field that the system updates with the contents of storage location X'30E' after the system is restarted. The operator is not required to supply any information but can store 1 byte of information into location X'30E' before initiating a restart. ACTCODE is valid only if you specify WAITTYPE=RESTARTABLE on the modify form of the macro.

,MF=(M,*list addr*)
Specifies the modify form of the LOADWAIT macro.

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

Example 1

Reserve storage for a parameter list named 'WAITPRM', then modify the existing list to load a wait state code and reason code. Assume that you will not know all the wait state information at program assembly time, so you must invoke LOADWAIT twice.

```
         LOADWAIT MF=(L,WAITPRM)
  .
  .
  .
         LOADWAIT WAITTYPE=NONREST,CODE=WAIT032,REASON=STOPCPU,
                     MF=(M,WAITPRM)
  .
  .
  .
WAIT032  DC   XL2'0032'            * WAIT STATE CODE IS KNOWN
STOPCPU  DS   XL2                  * STOPPED PROCESSOR IS NOT KNOWN
  .
  .
  .
```

Example 2

Reserve storage for a parameter list named 'WAITPRM', then modify the existing list to load a wait state code and reason code. Also specify 'OPRESP' as the action code and 'MOREINFO' for the PSAPARM. Assume that you will not know all the wait state information at program assembly time, so you must invoke LOADWAIT twice.

```
          LOADWAIT MF=(L,WAITPRM)
  .
  .
  .
          LOADWAIT WAITTYPE=RESTARTABLE,CODE=WAIT045,ACTCODE=OPRESP,
            PSAPARM=MOREINFO,REASON=REASON01,MF=(M,WAITPRM)
  .
  .
  .
WAIT045  DC   XL2'45'              * WAIT STATE CODE IS KNOWN
REASON01 DC   XL2'7A'              * REASON CODE IS KNOWN
MOREINFO DC   XL4'8007A045'        * PSAPARM IS KNOWN
OPRESP   DS   XL1                  * ACTION CODE ADDRESS IS NOT KNOWN
  .
  .
  .
```

LOCASCB — Locate Address Space Control Block (ASCB) Address

Description

The system identifies an address space through an address space identifier (ASID), an address space control block (ASCB), or a space token (STOKEN). Depending on the MVS service you want to use, you might be required to supply an identifier you do not have. For example, you might have the ASID of an address space but need to supply the ASCB. If you have the ASID or STOKEN but need to supply the ASCB, use the LOCASCB macro to return the ASCB address.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state or supervisor 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 or secondary when you specify the ASID parameter; primary or access register (AR) when you specify STOKEN.
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	CMS lock, if you want to be sure that the address space doesn't terminate while the system is referencing the ASCB; otherwise, no requirement.
Control parameters:	If you specify the ASID parameter, control parameters must be in the primary address space; if you specify the STOKEN parameter, 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

None.

Restrictions

None.

Input Register Information

Before issuing the LOCASCB 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

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

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

Register	Contents
0	Used as a work register by the system
1	ASCB address or 0
2-13	Unchanged
14	Used as a work register by the system

© Copyright IBM Corp. 1988, 2000

15	Return code

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

Register	Contents
0	Used as a work register by the system
1	0 if you specify STOKEN; otherwise, used as a work register by the system
2-13	Unchanged
14-15	Used as work registers by the system

Performance Implications

None.

Syntax

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

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

ASID=*asid addr*	*asid addr*: RX-type address or register (0) - (15).
STOKEN=*stoken addr*	*stoken addr*: RX-type address

Parameters

The parameters are explained as follows:

ASID=*asid addr*
Specifies the RX-type address of a halfword that contains the ASID for which the ASCB is to be returned or the register that contains the ASID in bits 16-31. (Bits 0-15 of the register are ignored.)

STOKEN=*stoken addr*
Specifies the RX-type address of the STOKEN that identifies the address space for which the ASCB is to be returned.

ABEND Codes

None.

Return Codes

When the LOCASCB macro returns control to your program, GPR 15 contains a hexadecimal return code.

Figure 9 (Page 1 of 2). Return Codes for the LOCASCB Macro

Return Code	Meaning and Action
00	**Meaning**: LOCASCB successfully located and returned the ASCB address. **Action**: None.

Figure 9 (Page 2 of 2). Return Codes for the LOCASCB Macro

Return Code	Meaning and Action
04	**Meaning**: Program error. The ASID or STOKEN did not map to a valid, active ASCB. This may occur because the input ASID of STOKEN never was valid, or because the address space it represents is no longer active. **Action**: Verify that the input ASID or STOKEN is valid.
08	**Meaning**: Program error. The STOKEN was not valid. **Action**: Supply a valid input STOKEN.

Example 1

Get the ASCB address for the address space whose ASID is specified by the constant at location ASN.

```
       LA    4,ASN
       LOCASCB ASID=(4)

ASN    DC    H'34'
```

Example 2

Get the ASCB address for the address space whose STOKEN is stored at the location STOKADDR.

```
         LOCASCB STOKEN=STOKADDR

STOKADDR DS    2F
```

LXFRE — Free a Linkage Index

Description

The LXFRE macro frees one or more linkage indexes. You cannot free a linkage index that was reserved with the SYSTEM option. (See the LXRES macro). Before issuing the LXFRE macro, disconnect all entry tables associated with the linkage index, unless you specify FORCE=YES. If you do not disconnect the entry tables and do not specify FORCE=YES, linkage indexes are not freed and the routine is abnormally terminated.

Related macro

LXRES

Environment

These are the requirements for the caller:

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

Programming Requirements

None.

Restrictions

None.

Input Register Information

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

Output Register Information

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

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

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

© Copyright IBM Corp. 1988, 2000

Performance Implications

None.

Syntax

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

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

LXLIST=*list addr*	*list addr*: RX-type address or register (0) - (12).
,FORCE=NO ,FORCE=YES	**Default**: FORCE=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

LXLIST=*list addr*
Specifies the address of a variable length list of fullword entries. The first word in the list must contain the number (1 to 32) of linkage indexes to be freed. Each entry following the first must contain a linkage index value specified in the form returned by the LXRES macro.

,FORCE=NO
,FORCE=YES
Specifies whether (YES) or not (NO) the linkage index is to be freed even if entry tables are currently connected to it. Any connected entry tables are disconnected before the linkage index is freed. FORCE=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 information specified can be any valid coding values.

ABEND Codes

052
053

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return Codes

When LXFRE macro returns control to your program, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code.

Figure 10. Return Codes for the LXFRE Macro	
Return Code	**Meaning and Action**
00	**Meaning**: The specified linkage indexes were freed. No entry tables were connected. **Action**: None.
04	**Meaning**: The specified linkage indexes were freed. Entry tables were connected, but FORCE was specified and was successfully executed. **Action**: None.
08	**Meaning**: Some of the specified linkage indexes were freed. Entry tables were connected. FORCE was specified but one or more of the necessary disconnects failed. **Action**: None required.

Examples

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in *OS/390 MVS Programming: Extended Addressability Guide*.

LXFRE—List Form

The list form of the LXFRE macro is used to construct a nonexecutable parameter list. The execute form of the LXFRE macro can refer to or modify the parameter list.

Syntax

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

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede LXFRE.
LXFRE	
␣	One or more blanks must follow LXFRE.
LXLIST=*list addr*	*list addr*: A-type address.
,FORCE=NO ,FORCE=YES	**Default**: FORCE=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

Parameters

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

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

LXFRE—Execute Form

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

Syntax

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

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

LXLIST=*list addr*	*list addr*: RX-type address or register (0) - (12).
,FORCE=NO ,FORCE=YES	**Default**: FORCE=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=(E,*cntl addr*)	*cntl addr*: RX-type address or register (0) - (12).

Parameters

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

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

LXRES — Reserve a Linkage Index

Description

The LXRES macro reserves one or more linkage indexes for the caller's use. The reserved linkage indexes are owned by the cross memory resource ownership task of the current home address space. The linkage index reservation applies across all linkage tables in the system and remains in effect until one of the following happens:

- An LXFRE macro explicitly frees a reserved linkage index.
- The cross memory resource ownership task terminates.
- The operator re-IPLs the system.

Related macro

LXFRE

Environment

These are the requirements for the caller:

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

Programming Requirements

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

Restrictions

None.

Input Register Information

Before issuing the LXRES 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

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

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

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

© Copyright IBM Corp. 1988, 2000

Performance Implications

None.

Syntax

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

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

LXLIST=*list addr*	*list addr*: RX-type address or register (0) - (12).
,SYSTEM=NO ,SYSTEM=YES	**Default**: SYSTEM=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

LXLIST=*list addr*
Specifies the address of a variable-length list of fullword entries. The first fullword in the list must contain the number (1 to 32) of linkage index values to be returned. The list must be long enough to contain the requested number of values. The linkage index values are returned in the list entries in the proper position for ORing with the entry index to form a PC number.

,SYSTEM=NO
,SYSTEM=YES
Specifies whether (YES) or not (NO) the linkage indexes are being reserved for system connections. If YES is specified, a subsequent ETCON macro specifying the linkage index causes all address spaces to be connected to the entry table.

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

ABEND Codes

052
053

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return Codes

When the LXRES macro returns control to your program, GPR 15 contains a hexadecimal return code.

Figure 11. Return Code for the LXRES Macro

Return Code	Meaning
00	**Meaning**: The specified linkage indexes were successfully reserved.

Examples

For examples of the use of this and other cross memory macros, refer to the chapter on cross memory communication in *OS/390 MVS Programming: Extended Addressability Guide*.

LXRES—List Form

The list form of the LXRES macro is used to construct a nonexecutable parameter list. The execute form of the macro can then refer to this list or a copy of it for reentrant programs.

Syntax

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

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

LXLIST=*list addr*	*list addr*: A-type address.
,SYSTEM=NO ,SYSTEM=YES	**Default**: SYSTEM=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

Parameters

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

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

LXRES—Execute Form

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

Syntax

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

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

LXLIST=*list addr*	*list addr*: RX-type address or register (0) - (12).
,SYSTEM=NO ,SYSTEM=YES	**Default**: SYSTEM=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=(E,*cntl addr*)	*cntl addr*: RX-type address or register (0) - (12).

Parameters

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

,MF=(E,*cntl addr)*

Specifies the execute form of the LXRES macro, and *cntl addr* is the name or address of the list form of the macro.

MCSOPER — Manage Extended MCS Operations

Description

MCSOPER enables you to activate and manage extended MCS consoles. An extended MCS console is a program that acts as a console. It can issue MVS commands, and receive command responses and unsolicited message traffic. MCSOPER defines and activates an extended MCS console to the system and provides a means of storing operator messages and command responses. MCSOPER also deactivates the extended console or console class, allows the extended console to receive the hardcopy message set, and releases migration IDs. See *OS/390 MVS Programming: Authorized Assembler Services Guide* for a definition of migration ID.

Use the MCSOPMSG macro to retrieve messages stored using MCSOPER. For more information on MCSOPER and MCSOPMSG, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0-7
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
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming Requirements

None.

Restrictions

In a JES3 environment prior to JES3 5.2.1 or in a mixed sysplex containing pre-MVS SP V5 systems, results might vary when you issue an MCSOPER ACTIVATE request with the HARDCOPY OPERPARM option. (See *OS/390 MVS Planning: Operations*.)

Input Register Information

Before issuing the MCSOPER macro, the caller must ensure that the following general purpose register (GPR) contains the specified information:

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

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	If GPR 15 contains a hexadecimal return code of 00, 10, or 14, GPR 0 contains a reason code; otherwise, GPR 0 contains 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 access registers (ARs) contain:

© Copyright IBM Corp. 1988, 2000

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

- MSGDLVRY=FIFO delivers better performance than MSGDLVRY=SEARCH. Use MSGDLVRY=SEARCH when you are looking only for certain types of messages such as command responses.
- If you request that an extended console receive the hardcopy message set from all the systems in a sysplex by specifying the MSCOPE=*ALL console attribute with the HARDCOPY OPERPARM option, you might experience performance problems.

Syntax

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

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

REQUEST=ACTIVATE REQUEST=DEACTIVATE REQUEST=RELEASE	See Figure 12 on page 59 for parameters available with REQUEST services.
,NAME=*opername addr*	*opername addr*: RX-type address or register (2) - (12).
,CONSID=*console id addr*	*console id addr*: RX-type address or register (2) - (12).
,ABTERM=NO ,ABTERM=YES	**Default**: ABTERM=NO
,TERMNAME=*terminal name addr*	*terminal name addr*: RX-type address or register (2) - (12).
,OPERPARM=*parm area addr*	*parm area addr*: RX-type address or register (2) - (12).
,MCSCSA=*csa addr*	*csa addr*: RX-type address or register (2) - (12).
,MCSCSAA=*alet addr*	*alet addr*: RX-type address or register (2) - (12).
,MSGDLVRY=FIFO ,MSGDLVRY=SEARCH ,MSGDLVRY=NONE	See Figure 13 on page 59 for parameters valid with MSGDLVRY services.
,MSGECB=*ecb addr*	*ecb addr*: RX-type address or register (2) - (12).
,QLIMIT=*qlimit addr*	*qlimit addr*: RX-type address or register (2) - (12). **Default**: QLIMIT=2147483647

,ALERTECB=*alert addr* ,ALERTECB=0	*alert addr*: Rx-type address or register (2) - (12). **Default**: ALERTECB=0
,ALERTPCT=*percent addr* ,ALERTPCT=*percent num*	*percent addr*: Rx-type address or register (2) - (12). *percent num*: number from 0 to 100. **Default**: ALERTPCT=100
,QRESUME=*qresume addr* ,QRESUME=*qresume num*	*qresume addr*: RX-type address or register (2) - (12). *qresume num*: number from 0 to 99. **Default**: QRESUME=0
,MIGID=*id addr*	*id addr*: RX-type address or register (2) - (12).
,MIGIDREL=*NO* ,MIGIDREL=*AUTO*	**Default**: MIGIDREL=NO.
,RTNCODE=*ret code*	*ret code*: RX-type address or register (2) - (12).
,RSNCODE=*reason code*	*reason code*: Rx-type address or register (2) - (12).

Figure 12. Parameters available with REQUEST= services

Parameters	REQUEST= ACTIVATE	REQUEST= DEACTIVATE	REQUEST= RELEASE
NAME	required	not valid	not valid
CONSID	required	required	not valid
ABTERM	not valid	optional	not valid
TERMNAME	required	not valid	not valid
OPERPARM	optional	not valid	not valid
MCSCSA	required	not valid	not valid
MCSCSAA	required	not valid	not valid
MSGDLVRY	optional	not valid	not valid
MIGID	optional	not valid	required
MIGIDREL	optional	not valid	not valid
RTNCODE	optional	optional	optional
RSNCODE	optional	optional	optional

Figure 13. Parameters available with MSGDLVRY= services

Parameters	MSGDLVRY= FIFO	MSGDLVRY= SEARCH	MSGDLVRY= NONE
MSGECB	required	required	not valid
QLIMIT	optional	optional	not valid
ALERTECB	optional	optional	not valid
ALERTPCT	optional	optional	not valid
QRESUME	optional	optional	not valid

Parameters

The parameters are explained as follows:

REQUEST=ACTIVATE
REQUEST=DEACTIVATE
REQUEST=RELEASE

Specifies the MCSOPER function you want to perform. This is a required parameter. The functions are as follows:

- ACTIVATE identifies an extended console to MCS. It initializes a specific session.
- DEACTIVATE identifies the console as inactive and terminates the session.

- RELEASE releases a migration ID that has been deactivated by issuing REQUEST=DEACTIVATE.

You can specify only one of these functions each time you invoke MCSOPER.

,NAME=*opername addr*
Specifies the address of an 8-byte field that contains the name of the console to be activated. NAME is required if you specified REQUEST=ACTIVATE. Do not use a name that might match the name of a device number (for example, BAD). If the name of a console matches a device number, a VARY command might not work as expected.

,CONSID=*console id addr*
Specifies the address of the 4-byte user console ID. If you specified REQUEST=ACTIVATE, CONSID is an output area that will receive the console ID. If you specified REQUEST=DEACTIVATE, supply the address of the ID you wish to deactivate. CONSID is required if you specified either REQUEST=ACTIVATE or REQUEST=DEACTIVATE.

,ABTERM=NO
,ABTERM=YES
Indicates an abnormal termination of an extended MCS console. If you request to deactivate a user console ID by abnormally terminating the console, the system switches the extended MCS console to an active alternate console, if one is available.

If ABTERM is not specified, ABTERM=NO is the default.

,TERMNAME=*terminal name addr*
Specifies the terminal name, VTAM LU name, or other identification of the source for the activate request. The name is logged when the activate occurs and has no other use.

,OPERPARM=*parm area addr*
Specifies the address of the MCSOP data area, mapped by IEZVG111. That area contains information on operator parameters such as routing codes, command authority, message format, message level, and whether the console will receive the hardcopy message set. OPERPARM is optional. The system looks for information on operator parameters first in the user profile of a security product, such as RACF. If the system does not find operator parameters in the user profile, it uses the operator parameters passed in the MCSOP data area, mapped by IEZVG111. If you did not specify the OPERPARM parameter, the system takes the default values of the operator parameters, also defined in the MCSOP data area. You can override the console attributes specified in the user profile of the security product by turning on bit MCSOVRDY in the MCSOP data area. For more information about OPERPARM, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

,MCSCSA=*csa addr*
Specifies the address of a 4-byte output area that will contain the address of the MCS console status area. When MCSOPER posts the ECB and if the return and reason codes indicate that queueing has been disabled, you need to check the MCS console status area, which is defined in the MCSCSA data area, mapped by IEAVG131, to determine which problem has disabled queuing. The MCSCSA parameter is required if you specified REQUEST=ACTIVATE.

,MCSCSAA=*alet addr*
Specifies the address of a 4-byte output area on a fullword boundary that will contain the ALET that identifies the address or data space that contains the MCS console status area. MCSCSAA is required if you specified REQUEST=ACTIVATE.

,MSGDLVRY=FIFO
,MSGDLVRY=SEARCH
,MSGDLVRY=NONE
Specifies how MCSOPER queues messages to extended MCS consoles. Specifying FIFO will cause MCSOPER to deliver messages on a first-in, first-out basis. This is the default option. Specifying SEARCH allows you to request messages based on search

arguments specified in the MCSOPMSG macro. Specifying NONE will keep messages from being delivered to this console.

,MSGECB=*ecb addr*
Specifies the address of the ECB that the system posts when it queues a message to the console. Note that the system posts this ECB for every message it delivers to the message dataspace. However, since it is not possible to tell how many times an ECB has been posted since it was last waited on, once an ECB is posted, users should issue MCSOPMSG to retrieve messages until they receive a return and reason code indicating that no more messages remain to be retrieved. This keyword is optional and is not valid if you specified MSGDLVRY=NONE.

,QLIMIT=*qlimit addr*
Specifies either the maximum number or the address of the maximum number of messages which the system can queue to the console issuing the request. You can specify the maximum number as any number from 1 to 2147483647 (2 gigabytes). QLIMIT=2147483647 is the default.

,ALERTECB=*alert addr*
,ALERTECB=0
Specifies the address of the ECB that the system will post when one of the following is true:

- The number of messages in storage has reached the number specified in QLIMIT.
- The number of messages in storage matches the percentage of the QLIMIT number specified in ALERTPCT.
- You ran out of space for messages in storage.
- There is a queuing error.

If you specify ALERTECB=0, MCSOPER will not post an ECB.

,ALERTPCT=*percent addr*
,ALERTPCT=*percent num*
Specifies the address of the percentage of the maximum number of messages specified in QLIMIT or the number representing the percent. When the queue reaches this percentage, the MCSOPER posts the ALERTECB. If you do not specify *percent addr*, or *percent num* is outside the range of 1 through 99, MCSOPER will operate on the default, 100.

,QRESUME=*qresume*
,QRESUME=*qresume num*
Specifies the address that contains the depth percent at which the queue must be to resume queuing. If the system disabled queuing because the number of messages in the data space reached the percentage of the QLIMIT (ALERTPCT) and you retrieve enough messages to meet the percentage you specify in QRESUME, queuing will resume automatically for the console issuing the request. If you do not specify QRESUME, or QRESUME is outside the range of 0 through 99, queuing will only resume after explicit action by the application using the MCSOPMSG REQUEST=RESUME macro. The default is QRESUME=0.

,MIGID=*id addr*
Specifies the address of a one-byte output and input area. *id addr* specifies either the address of the area in which MCSOPMSG returns the migration ID assigned when REQUEST=ACTIVATE is specified or the address of the name of the migration ID you wish to release, when specified with REQUEST=RELEASE. This ID must be in the range of 100 through 127 or 129 through 250.

,MIGIDREL=NO.
,MIGIDREL=AUTO.
Specifies if the system will automatically release the migration ID. If NO is specified, the migration ID will never be automatically released. This is the default option. If AUTO is specified, the migration ID will be automatically released when a requested OR

system-initiated deactivation occurs. This parameter is ignored if a migration ID is not requested.

Note that when you issue MCSOPER REQUEST=ACTIVATE with the optional MIGIDREL=AUTO parameter, you must not issue MCSOPER REQUEST=RELEASE following the MCSOPER REQUEST=DEACTIVATE anymore (since the migration ID is automatically released as a result of the deactivation).

If MIGIDREL is not specified, MIGIDREL=NO is the default.

,RTNCODE=*ret code*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15.

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

ABEND Codes

None.

Return and Reason Codes

When control returns from MCSOPER, GPR 15 (and *ret code*, if you specified RTNCODE) contains one of the following hexadecimal return codes. GPR 0 (and *reason code*, if you specified RSNCODE) contains one of the following hexadecimal reason codes.

Figure 14 (Page 1 of 3). Return and Reason Codes for the MCSOPER Macro

Return Code	Reason Code	Meaning and Action
00	00	**Meaning**: Processing was successful. **Action**: None.
00	04	**Meaning**: Environmental error. MCSOPER did not obtain a migration ID if you requested one. **Action**: If you need a migration ID, deactivate a console and retry the request. It might be necessary to release some other migration IDs.
04	None	**Meaning**: Environmental error. For REQUEST=ACTIVATE, the console was already active. For REQUEST=DEACTIVATE, the console was already inactive. **Action**: If you specified the wrong console, correct the error and retry the request.
08	None	**Meaning**: Program or environmental error. For REQUEST=DEACTIVATE, the console has not been defined. **Action**: If you specified the wrong console, correct the error and retry the request. If the console should have been active, find out why it was not, correct the problem, and rerun the program.
0C	None	**Meaning**: For an ACTIVATE request, the issuer does not have read authority access to resource name, MVS.MCSOPER.*, where * is the name of the console. **Action**: Correct the problem and rerun the program.
10	00	**Meaning**: System error. The input parameter list contains an error. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.
10	08	**Meaning**: System error. The input parameter list contains an error. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.

Figure 14 (Page 2 of 3). Return and Reason Codes for the MCSOPER Macro

Return Code	Reason Code	Meaning and Action
10	0C	**Meaning**: Program error. The specified console ID is not valid. The reason could be one of the following: • The syntax of the console ID is incorrect • You specified the wrong console ID. **Action**: Correct any errors and rerun the program.
10	18	**Meaning**: Program error. The authority level specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	1C	**Meaning**: Program error. The message format specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	20	**Meaning**: Program error. The message level specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	24	**Meaning**: Program error. The message type specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	28	**Meaning**: Program error. The log command response specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	2C	**Meaning**: System error. This reason code is used for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.
10	30	**Meaning**: Program error. The key specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	34	**Meaning**: Program error. The specified migration ID is not valid. **Action**: To determine the valid migration IDs, issue the DISPLAY C command. Obtain the correct value, correct the problem, and rerun the program.
10	38	**Meaning**: Program error. The command association specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	3C	**Meaning**: Program error. The message scope specified in the OPERPARM segment is not valid. One of the following errors exists: • The calling program specified ALL with some other value • A system name contains a syntax error • A duplicate system name exists in the list of systems • The specified number of systems is not valid. **Action**: Correct the problem and rerun the program.
10	40	**Meaning**: Program error. The alternate group name specified in the OPERPARM segment is not valid. **Action**: Correct the problem and rerun the program.
10	44	**Meaning**: You issued MCSOPER while in cross-memory mode. You cannot issue MCSOPER if your program is in cross-memory mode. **Action**: Correct the problem and rerun the program.
14	00	**Meaning**: System error. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.
14	08	**Meaning**: System error. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.

Figure 14 (Page 3 of 3). Return and Reason Codes for the MCSOPER Macro

Return Code	Reason Code	Meaning and Action
14	0C	**Meaning**: Program or environmental error. There was an SAF routine error. **Action**: Check your OPERPARM statement for incorrect entries. If you do not find an error, record the return and reason code and supply them to the appropriate IBM support personnel.
14	10	**Meaning**: System error. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.
14	14	**Meaning**: System error. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.
14	18	**Meaning**: Program error. The console ID you specified is not defined to the sysplex at this time. **Action**: Check to see that you specified the correct console ID. Determine whether the console is active by issuing a DISPLAY C command from a console. Correct the problem and rerun the program.
14	1C	**Meaning**: System error. This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.
14	20	**Meaning**: System error. There was a data space initialization error. The system could not create a data space because it could not obtain a data space or ALET. **Action**: This might be a performance or tuning problem. Contact your system programmer.
14	24	**Meaning**: This reason code is for IBM diagnostic purposes only. **Action**: Record the return and reason code and supply them to the appropriate IBM support personnel.
14	28	**Meaning**: System error. Necessary storage could not be obtained for the console. **Action**: Attempt to rerun the program. If the error persists, record the return and reason code and supply them to the appropriate IBM support personnel.
18	None	**Meaning**: Program error. The caller is not in supervisor state. **Action**: Correct your program and resubmit it.
1C	None	**Meaning**: Environmental error. For REQUEST=RELEASE, the user identified by the migration ID is not defined to the system. **Action**: Determine the correct migration ID and rerun the program.
20	None	**Meaning**: Environmental error. For REQUEST=RELEASE, the console with the specified migration ID is still active. **Action**: Make sure you specified the correct migration ID. If so, deactivate the console and reissue REQUEST=RELEASE.

Example 1

This example activates a console named TAPE1 with operator parameter information contained in an area called PARMAREA, and values for MCSCSA and return and reason codes.

```
MDR      CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
   MCSOPER REQUEST=ACTIVATE,NAME=TAPE1,CONSID=IDAREA,                  X
               MSGECB=ALERT_ECB,TERMNAME=TERMINAL,                     X
               OPERPARM=PARMAREA,                                      X
               MCSCSA=STATUS_AREA,MCSCSAA=MY_ALET,                     X
               RTNCODE=RETCODE,RSNCODE=REASON
         LM    14,12,12(13)
         BR    14
*
PARMAREA DS    CL40
IDAREA   DS    CL4
ASCBPTR  DS    A
CLASSNAME DS   CL8
TERMINAL  DC    CL8'CN3E0'
TAPE1    DC    CL8'TAPE1'
RETCODE  DS    F
REASON   DS    F
ALERT_ECB DS  A
          DS  0D
STATUS_AREA DS   CL4
MY_ALET   DS  F
         IEZVG111
         END   MDR
```

Example 2

This example deactivates a console named TAPE1 and switches processing to an active alternate console.

```
MDR      CSECT
         STM   14,12,12(13)
         BALR  12,0
         USING *,12
   MCSOPER REQUEST=DEACTIVATE,CONSID=IDAREA,ABTERM=YES,                X
               RTNCODE=RETCODE,RSNCODE=REASON
         LM    14,12,12(13)
         BR    14
*
IDAREA   DS    CL4
RETCODE  DS    F
REASON   DS    F
         END   MDR
```

Example 3

Activate an extended MCS console and specify that it is to receive the hardcopy message set.

```
TESTHC   CSECT
TESTHC   AMODE 31
TESTHC   RMODE ANY
*
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
```

```
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
         STM   R14,R12,12(R13)
         BALR  R12,0
         USING *,R12
         MODID BRANCH=YES
*
         GETMAIN RU,LV=DATAEND        Obtain storage for data areas
         LR    R11,R1
         USING DATAAREA,R11
         ST    R13,SAVEAREA+4
         LA    R15,SAVEAREA
         ST    R15,8(R13)
         LR    R13,R15
*
         LA    R8,OPERDATA            Address of operparm area
         USING MCSOPPRM,R8            IEZVG111 addressability
         XC    MCSOPPRM(MCSOPLEN),MCSOPPRM  Clear the operparm area
*************************************************************************
* Override the console attributes specified in the user profile
* of the security product by turning on bit MCSOVRDY in the MCSOP data
* area. Request the hardcopy attribute (to receive hardcopy message set).
*************************************************************************
         OI    MCSOFLAG,MCSOVRDY      Override console attributes
         OI    MCSOMISC,MCSOHDCY      Request the hardcopy attribute
*
         MODESET MF=(E,SUP)           Change to supervisor state
*                                     to issue MCSOPER ACTIVATE request
*************************************************************************
* Activate an extended MCS console whose name is contained in a field
* called HCCONSNM.  The attributes of the extended MCS console are
* contained in a field called OPERDATA, mapped by IEZVG111.  The
* console will have its messages delivered on a first-in-first-out
* basis.  The system will post a message ECB called HCMECB.
* The address of the output area that contains the
* address of the MCS console status area is contained in a field
* called HCSTATUS.  The address of the ALET that identifies the address
* or data space that contains the MCS console status area is
* contained in a field called HCSTATAL.
* The system returns the console ID in the field called HCCONSID.
* The system returns a return code and a reason code in fields
* called HCRETC and HCRNC, respectively.
*************************************************************************
```

```
         MCSOPER REQUEST=ACTIVATE,  Activate the console             X
               NAME=HCCONSNM,                                        X
               TERMNAME=HCCONSNM,                                    X
               OPERPARM=OPERDATA,                                    X
               MSGDLVRY=FIFO,                                        X
               MSGECB=HCMECB,                                        X
               MCSCSA=HCSTATUS,                                      X
               MCSCSAA=HCSTATAL,                                     X
               CONSID=HCCONSID,                                      X
               RTNCODE=HCRETC,                                       X
               RSNCODE=HCRSNC,                                       X
               MF=(E,MCSOPPL)
*
         MODESET MF=(E,PROB)       Return to problem state
*
         L     R13,4(R13)
         FREEMAIN RU,LV=DATAEND,A=(R11),SP=230
         LM    R14,R12,12(R13)
         BR    R14
*
HCCONSNM DC    CL8'HCMCSOP '
SUP      MODESET MODE=SUP,MF=L     Parmlist for supervisor state
PROB     MODESET MODE=PROB,MF=L    Parmlist for problem state
*
DATAAREA DSECT
         DS    0F
SAVEAREA DS    18F
         DS    0F
OPERDATA DS    CL(MCSOPLEN)
HCCONSID DS    CL4
HCSTATUS DS    A
HCSTATAL DS    F
HCMECB   DS    F
HCRETC   DS    F
HCRSNC   DS    F
         MCSOPER MF=(L,MCSOPPL)
DATAEND  EQU   *-DATAAREA
         IEZVG111
         END   TESTHC
```

MCSOPER—List Form

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

name	*name*: Symbol. Begin *name* in column 1.
blank;	One or more blanks must precede MCSOPER.
MCSOPER	
ъ	One or more blanks must follow MCSOPER.

,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 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 MCSOPER 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.

MCSOPER—Modify Form

The modify form of the MCSOPER macro changes parameters in the control parameter list that the system created through the list form of the macro.

Syntax

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

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

REQUEST=ACTIVATE REQUEST=DEACTIVATE REQUEST=RELEASE	See Figure 12 on page 59 for parameters available with REQUEST services.
,NAME=*opername addr*	*opername addr*: RX-type address or register (2) - (12).
,CONSID=*console id addr*	*console id addr*: RX-type address or register (2) - (12).
,ABTERM=NO ,ABTERM=YES	**Default**: ABTERM=NO
,TERMNAME=*terminal name addr*	*terminal name addr*: RX-type address or register (2) - (12).
,OPERPARM=*parm area addr*	*parm area addr*: RX-type address or register (2) - (12).
,MCSCSA=*csa addr*	*csa addr*: RX-type address or register (2) - (12).
,MCSCSAA=*alet addr*	*alet addr*: RX-type address or register (2) - (12).
,MSGDLVRY=FIFO ,MSGDLVRY=SEARCH ,MSGDLVRY=NONE	See Figure 13 on page 59 for parameters valid with MSGDLVRY services.

,MSGECB=*ecb addr*	*ecb addr*: RX-type address or register (2) - (12).
,QLIMIT=*qlimit addr*	*qlimit addr*: RX-type address or register (2) - (12). **Default**: QLIMIT=2147483647
,ALERTECB=*alert addr* ,ALERTECB=0	*alert addr*: Rx-type address or register (2) - (12). **Default**: ALERTECB=0
,ALERTPCT=*percent addr* ,ALERTPCT=*percent num*	*percent addr*: Rx-type address or register (2) - (12). *percent num*: Number from 0 to 100. **Default**: ALERTPCT=100
,QRESUME=*qresume addr* ,QRESUME=*qresume num*	*qresume addr*: RX-type address or register (2) - (12). *qresume num*: Number from 0 to 99. **Default**: QRESUME=0
,MIGID=*id addr*	*id addr*: RX-type address or register (2) - (12).
,MIGIDREL=*NO* ,MIGIDREL=*AUTO*	**Default**: MIGIDREL=NO.
,RTNCODE=*ret code*	*ret code*: RX-type address or register (2) - (12).
,RSNCODE=*reason code*	*reason code*: Rx-type address or register (2) - (12).
,MF=(M,*list addr*,COMPLETE) ,MF=(M,*list addr*,NOCHECK)	*list addr*: RX-type address or register (2) - (12). **Default**: COMPLETE

Parameters

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

,MF=(M,*list addr***,COMPLETE)**
,MF=(M,*list addr***,NOCHECK)**
Specifies the modify form of the MCSOPER macro.

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

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that are not specified. NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify.

MCSOPER—Execute Form

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

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

REQUEST=ACTIVATE	See Figure 12 on page 59 for parameters available with REQUEST services.
REQUEST=DEACTIVATE REQUEST=RELEASE	
,NAME=*opername addr*	*opername addr*: RX-type address or register (2) - (12).
,CONSID=*console id addr*	*console id addr*: RX-type address or register (2) - (12).
,ABTERM=NO ,ABTERM=YES	**Default**: ABTERM=NO
,TERMNAME=*terminal name addr*	*terminal name addr*: RX-type address or register (2) - (12).
,OPERPARM=*parm area addr*	*parm area addr*: RX-type address or register (2) - (12).
,MCSCSA=*csa addr*	*csa addr*: RX-type address or register (2) - (12).
,MCSCSAA=*alet addr*	*alet addr*: RX-type address or register (2) - (12).
,MSGDLVRY=FIFO	See Figure 13 on page 59 for parameters valid with MSGDLVRY services.
,MSGDLVRY=SEARCH ,MSGDLVRY=NONE	
,MSGECB=*ecb addr*	*ecb addr*: RX-type address or register (2) - (12).
,QLIMIT=*qlimit addr*	*qlimit addr*: RX-type address or register (2) - (12). **Default**: QLIMIT=2147483647
,ALERTECB=*alert addr* ,ALERTECB=0	*alert addr*: Rx-type address or register (2) - (12). **Default**: ALERTECB=0
,ALERTPCT=*percent addr* ,ALERTPCT=*percent num*	*percent addr*: Rx-type address or register (2) - (12). *percent num*: Number from 0 to 100. **Default**: ALERTPCT=100
,QRESUME=*qresume addr* ,QRESUME=*qresume num*	*qresume addr*: RX-type address or register (2) - (12). *qresume num*: Number from 0 to 99. **Default**: QRESUME=0
,MIGID=*id addr*	*id addr*: RX-type address or register (2) - (12).
,MIGIDREL=*NO* ,MIGIDREL=*AUTO*	**Default**: MIGIDREL=NO.
,RTNCODE=*ret code*	*ret code*: RX-type address or register (2) - (12).
,RSNCODE=*reason code*	*reason code*: Rx-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE) ,MF=(E,*list addr*,NOCHECK)	*list addr*: RX-type address or register (2) - (12). **Default**: COMPLETE

Parameters

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

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

Specifies the execute form of the MCSOPER macro.

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

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that are not specified. NOCHECK specifies

that the system does not check for required parameters and does not supply the optional parameters that you did not specify.

MCSOPMSG — Retrieve MCS Operator Messages

Description

MCSOPMSG retrieves messages queued to an extended MCS console, and returns message information in a message data block (MDB). "Retrieving" a message more specifically means that MCSOPMSG returns the address of the MDB that contains the message and information pertaining to that message. MCSOPMSG also resumes queuing to a message storage area if queuing was previously suspended.

Use MCSOPMSG to retrieve messages stored using the MCSOPER macro. For more information on MCSOPMSG, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

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

Programming Requirements

- The console for which you are issuing this macro must already be activated.
- If you specified MSGDLVRY=NONE on the MCSOPER macro for a particular console, do not issue MCSOPMSG for that console.
- Only the address space that activated the extended console can issue MCSOPMSG for that console.
- If you specified MSGDLVRY=FIFO for MCSOPER and you are running in a multi-task environment, ensure that only one task is currently performing MCSOPMSG for the console at any time.
- You can use the CMDRESP, CART, and MASK parameters only if you activate the console with MCSOPER MSGDLVRY=SEARCH.
- You must include mapping macro IEAVM105. You can use mapping macro IEAVM105 to access fields in the message data block (MDB), which is returned by MCSOPMSG. Refer to *OS/390 MVS Data Areas, Vol 3 (IVT-RCWK)* for the format and content of this information.

Restrictions

None.

Input Register Information

Before issuing the MCSOPMSG 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, 2000

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	If GPR 15 contains a return code of 08, 10, or 14, GPR 0 contains a reason code; otherwise, GPR 0 contains zero.
1	Address of the MDB if REQUEST=GETMSG was specified and the return code is zero or four. Otherwise, GPR 1 is used as a work register by the system.
2-14	Unchanged
15	Return code

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

Register	Contents
0	Used as a work register by the system
1	If your program is in AR mode and GPR 1 contains the address of an MDB, AR 1 contains the ALET for the MDB; otherwise, AR 1 is used as work register by the system
2-14	Unchanged
15	Used as a work register by the system

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

Performance Implications

None.

Syntax

The standard form MCSOPMSG macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede MCSOPMSG.
MCSOPMSG	
␣	One or more blanks must follow MCSOPMSG.
REQUEST=GETMSG	See Figure 15 on page 75 for parameters valid with REQUEST services.
REQUEST=RESUME	
,CMDRESP=NO ,CMDRESP=YES	**Default**: CMDRESP=NO
,CART=*cart addr*	*cart addr*: RX-type address or register (2) - (12).
,MASK=*mask addr*	*mask addr*: RX-type address or register (2) - (12).
,CONSID=*console id addr*	*console id addr*: RX-type address or register (2) - (12).
,RTNCODE=*return code addr*	*return code addr*: RX-type address or register (2) - (12).
,RSNCODE=*reason code addr*	*reason code addr*: RX-type address or register (2) - (12).

Figure 15. Parameters valid with REQUEST= services

Parameters	REQUEST=GETMSG	REQUEST=RESUME
CMDRESP	optional	not valid
CART	optional	not valid
MASK	optional	not valid
CONSID	required	required
RTNCODE	optional	optional
RSNCODE	optional	optional

Parameters

The parameters are explained as follows:

REQUEST=GETMSG
REQUEST=RESUME
Specifies the MCSOPMSG function you want to perform.

GETMSG retrieves a queued message from storage, and RESUME resumes message queuing.

,CMDRESP=NO
,CMDRESP=YES
Specifies the type of message search. NO indicates that MCSOPMSG will obtain the next message. YES indicates that MCSOPMSG will return the next command response message. CMDRESP is valid only with REQUEST=GETMSG, and is meaningful only if you specified MSGDLVRY=SEARCH on the MCSOPER macro.

,CART=*cart addr*
Specifies the address of an 8-character field that contains the name of the CART (command and response token) to use to search for the next message. A CART associates a command with a response. You can specify CART only if you specify CMDRESP=YES and you specified MSGDLVRY=SEARCH on MCSOPER.

,MASK=*mask addr*
Specifies the address of an 8-character field that contains the mask that MCSOPMSG will compare with the CART using the logical AND instruction. Specify MASK only if you specified CART.

,CONSID=*console id addr*
Specifies the address of the 4-byte console ID.

,RTNCODE=*return code addr*
Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,RSNCODE=*reason code 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 control returns from MCSOPMSG, GPR 15 (and *return code addr*, if you coded RTNCODE) contains one of the following hexadecimal return codes and GPR 0 (and *reason code addr*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Figure 16 (Page 1 of 2). Return and Reason Codes for the MCSOPMSG Macro

Return Code	Reason Code	Meaning and Action
00	None	**Meaning**: MCSOPMSG completed successfully. If REQUEST=GETMSG was specified, MCSOPMSG retrieved a message; if REQUEST=RESUME was specified, message queuing resumed. **Action**: None
04	None	**Meaning**: Program error. MCSOPMSG retrieved a message, but either message queuing is disabled (if you specified REQUEST=GETMSG) or message queuing was already enabled (if you specified REQUEST=RESUME). **Action**: For REQUEST=GETMSG, continue until all messages have been retrieved. Then, attempt a REQUEST=RESUME to allow the system to send messages to that console again.
08	00	**Meaning**: Program error. For REQUEST=GETMSG, MCSOPMSG attempted to retrieve a message, but there are no messages available for the specified search criteria. Note that this is the normal, expected, return and reason code issued when MCSOPMSG retrieved all messages currently queued to this EMCS console. **Action**: None if all messages have been retrieved or change the search criteria and reissue MCSOPMSG.
08	01	**Meaning**: Environmental error. For REQUEST=RESUME, queueing is not resumed because the message queue is being rebuilt. **Action**: If you specified ALERTECB=*alert address* on the MCSOPER request to define this extended MCS console, the calling program will receive control when the queue has been rebuilt. Either add this parameter to the MCSOPER specification and provide an ECB to be posted, or retry the request at a later time.
0C	None	**Meaning**: Program error or environmental error. MCSOPMSG did not retrieve a message. Message queuing is disabled. You either specified search criteria on the CMDRESP, CART, or MASK parameters for which there is no current match, or the extended console needs to be reset. **Action**: If you specified search criteria on REQUEST=GETMSG, specify different search criteria, or specify only the CONSID parameter. If you specified only the CONSID parameter, issue REQUEST=RESUME on the MCSOPMSG macro.
10	01	**Meaning**: Program error. The specified console is not active. **Action**: Verify that you specified the correct console. If so, take steps to activate the console and retry the request. If not, correct the error and retry the request.
10	02	**Meaning**: Program or environmental error. The specified console was not activated by this address space. **Action**: Ensure that you specified the correct console and that you issued MCSOPMSG from the correct address space. Correct the problem and retry the request.
14	01	**Meaning**: Program error. The parameter list contains an incorrect acronym or an incorrect version indicator. **Action**: Correct the problem and retry the request.
14	02	**Meaning**: Program error. The caller is not in AR mode. **Action**: Correct the problem and retry the request.

Figure 16 (Page 2 of 2). Return and Reason Codes for the MCSOPMSG Macro

Return Code	Reason Code	Meaning and Action
14	03	**Meaning**: Environmental error. Another MCSOPMSG request is in progress for this console. **Action**: This duplication can happen only if you specified MSGDLVRY=FIFO on the initial MCSOPER request. Either wait for the current request to complete and then retry, or do not specify MSGDLVRY=FIFO when you issue the MCSOPER macro.
14	04	**Meaning**: Program error. The extended console was activated with MSGDLVRY=FIFO, but MCSOPMSG was issued with the CMDRESP parameter. **Action**: Issue MCSOPMSG without the CMDRESP parameter.
14	05	**Meaning**: Program error. The caller is not in supervisor state. **Action**: Reissue MCSOPMSG in supervisor state.
18	None	**Meaning**: System error. The service ended abnormally. **Action**: Record the return code and supply it to the appropriate IBM support personnel.

Example

Obtain a message for a console whose ID is specified in the field named ID. Request message queuing to resume using a parameter list named DATA. The parameter list was created using the list form of the macro.

```
MDR      CSECT
R12      EQU   12
R13      EQU   13
R14      EQU   14
         STM   R14,R12,12(R13)                              0000600
         BALR  R12,0
         USING *,R12
         MCSOPMSG REQUEST=RESUME,CONSID=ID,MF=(E,DATA,COMPLETE)
         LM    R14,R12,12(R13)
         BR    R14
         DROP  R12
*********************************************************************
*        CONSTANTS AND DATA                                         *
*********************************************************************
ID       DS    F
         MCSOPMSG MF=(L,DATA)
         IEAVM105
         END   MDR
```

MCSOPMSG—List Form

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

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

,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 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 MCSOPMSG 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.

MCSOPMSG—Execute Form

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

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

REQUEST=RESUME	See Figure 17 on page 79 for parameters valid with REQUEST services.
REQUEST=GETMSG	
,CMDRESP=NO ,CMDRESP=YES	**Default**: CMDRESP=NO
,CART=*cart addr*	*cart addr*: RX-type address or register (2) - (12).
,MASK=*mask addr*	*mask addr*: RX-type address or register (2) - (12).
,CONSID=*console id addr*	*console id addr*: RX-type address or register (2) - (12).
,RTNCODE=*return code addr*	*return code addr*: RX-type address or register (2) - (12).
,RSNCODE=*reason code addr*	*reason code addr*: RX-type address or register (2) - (12).

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

Figure 17. Parameters valid with REQUEST= services for the execute form of the macro

Parameters	REQUEST=GETMSG	REQUEST=RESUME
CMDRESP	optional	not valid
CART	optional	not valid
MASK	optional	not valid
CONSID	required	required
RTNCODE	optional	optional
RSNCODE	optional	optional
MF	required	required

Parameters

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

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

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

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.

NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify.

MCSOPMSG—Modify Form

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

Syntax

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

name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede MCSOPMSG.
MCSOPMSG	
ƀ	One or more blanks must follow MCSOPMSG.
REQUEST=GETMSG	See Figure 18 on page 80 for parameters valid with REQUEST services.
REQUEST=RESUME	
,CMDRESP=NO ,CMDRESP=YES	**Default**: CMDRESP=NO
,CART=*cart addr*	*cart addr*: RX-type address or register (2) - (12).

,MASK=*mask addr*	*mask addr*: RX-type address or register (2) - (12).
,CONSID=*console id addr*	*console id addr*: RX-type address or register (2) - (12).
,RTNCODE=*return code addr*	*return code addr*: RX-type address or register (2) - (12).
,RSNCODE=*reason code addr*	*reason code addr*: RX-type address or register (2) - (12).
,MF=(M,*list addr*,COMPLETE) ,MF=(M,*list addr*,NOCHECK)	*list addr*: RX-type address or register (2) - (12). **Default**: COMPLETE

Figure 18. Parameters valid with REQUEST= services for the modify form of the macro

Parameters	REQUEST=GETMSG	REQUEST=RESUME
CMDRESP	optional	not valid
CART	optional	not valid
MASK	optional	not valid
CONSID	required	required
RTNCODE	optional	optional
RSNCODE	optional	optional
MF	required	required

Parameters

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

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

Specifies the modify form of the MCSOPMSG macro.

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

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.

NOCHECK specifies that the system does not check for required parameters and does not supply the optional parameters that you did not specify.

MGCR — Issue an Internal START or REPLY Command

Description

Note: IBM recommends that you use the MGCRE macro rather than MGCR.

The MGCR macro starts a program or subsystem from within your program and passes 31 bits of information, in the form of a token, to the started program. The MGCR macro can also issue a reply to a WTOR macro. In other words, use MGCR to issue an internal START or REPLY command.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0-7
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

- The command buffer can be located in 24-bit or 31-bit addressable storage.
- A program token is meaningful only with the START command.

Restrictions

You can use MGCR to issue only START or REPLY commands. You must use MGCRE for any other commands.

Input Register Information

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

Register	Contents
0	Zero
1	A pointer to a parameter list mapped by IEZMGCR.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Unchanged
1	Used as a work register by the system
2-14	Unchanged
15	For the START command, GPR 15 contains a return code; otherwise, GPR 15 is 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.

© Copyright IBM Corp. 1988, 2000

Performance Implications

None.

Syntax

The MGCR macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede MGCR.
MGCR	
ƀ	One or more blanks must follow MGCR.
command-buffer-address	*command-buffer-address*: RX-type address or register (1) or (2) - (12).

Parameters

The parameters are explained as follows:

command-buffer-address
Specifies the address of a command buffer mapped by the IEZMGCR macro. The command buffer must contain the following information:

Name	Length	Contents
flags1	1 byte	If bit 0 is one, then flags2 must contain meaningful information. Bits 1-7 must be zero.
length	1 byte	Length of the buffer up to but not including the program token field.
flags2	2 bytes	X‘0000’ - neither a program token nor a user security token are present. X‘0800’ - a program token is present. X‘0008’ - a user security token is present. X‘0808’ - both a program token and a user security token are present.
text	up to 126 bytes	Command, operands, and optional comments as follows: command operands comments
ptoken	31 bits right-justified	An optional field containing any desired information, such as an identifier that indicates the issuing program.
utoken	80 bytes	Indicates which user security token the system takes to use for a command issued from an MCS console. The possibilities are console, CTAS, *FAIL, or “undefined-user” ACEE.

ABEND Codes

MGCR might abnormally terminate with abend code X'D22'. See *OS/390 MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

Register 15 contains one of the following hexadecimal return codes as the result of a START command. No return codes result from the REPLY command.

Figure 19. Return Codes for the START Command

Return Code	Meaning and Action
00	**Meaning**: The START command processed successfully. Register 0 contains the right-justified ASID of the started address space. **Action**: None.
08	**Meaning**: Environmental error. The START command failed for one of the following reasons: • The START command specified a console that is not authorized for entering the command • The system did not allow the address space to be created at this time due to a heavy system workload • There is not enough storage available to schedule the command • The system tried to obtain more address spaces than the maximum number supported. **Action**: Check to see if the START command specified a console that is not authorized for entering the command, and correct the situation if necessary. Next, retry the request. If the problem persists, record the return code and supply it to the appropriate IBM support personnel.

Example

Issue an internal REPLY command in response to an action message. Security tokens are not in use.

```
ISSUMGCR EQU   *
         XC    MGCRPL(MGCRLTH),MGCRPL   Clear the parameter list
         MVC   MGCRTEXT(L'TXTINSRT),TXTINSRT   Move in the reply buffer
         MVC   REPLY,CTXTRPID           Insert the reply ID
         LA    REG1,(MGCRTEXT-MGCRPL)+L'TXTINSRT   Get MGCRPL length
         STC   REG1,MGCRLGTH            Save length in the MGCRPL
         SR    REG0,REG0                Clear register zero
         MGCR  MGCRPL                   Issue the command
         .
         .
         .
         IEZMGCR  DSECT=NO              Mapping of MGCR parameter list
         ORG   MGCRTEXT
COMMAND  DS    CL6                      Storage for REPLY verb
REPLY    DS    CL2                      Reply ID
REPLYMSG DS    CL3                      WTOR response
         ORG
```

MGCRE — Issue Internal Commands

Description

MGCRE allows a program to issue commands without operator intervention. For example, an application could issue a VARY or CONTROL command by using MGCRE, which might satisfy an outstanding action message.

MGCRE can pass a UTOKEN to SAF (Security Authorization Facility) to validate the authority of the issuer.

Note: You can still use the MGCR macro to issue internal START or REPLY commands. However, IBM recommends using MGCRE.

See *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information on using the MGCRE macro.

MGCRE has a list and an execute form, but no standard form.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0-7
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	Any
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

- It is possible that a console associated with a command has a CMDSYS parameter on a control command in effect. This condition may cause the command to be sent to another system in a sysplex.
- If you issue MGCRE from a program or an address space controlling a console, CMDSYS takes effect.
- The caller cannot have an EUT FRR established.

Input Register Information

Before issuing the MGCRE 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. If register 15 contains a return code of 0, register 0 contains the ASID of the started address space.
2-13	Unchanged
14	Used as a work register by the system

© Copyright IBM Corp. 1988, 2000

15 Used as a work register by the system, unless MGCRE entered a START command. In that case, register 15 contains a return code.

Performance Implications

None.

MGCRE—List Form

Use the list form of the MGCRE macro together with the execute form of the macro. 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 MGCRE macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede MGCRE.
MGCRE	
␣	One or more blanks must follow MGCRE.
,MF=L	

Parameters

The parameters are explained as follows:

,MF=L
Specifies the list form of MGCRE. Do not specify any other parameters with the list form of MGCRE.

MGCRE—Execute Form

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

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede MGCRE.
MGCRE	
␣	One or more blanks must follow MGCRE.
TEXT=*text addr*	*text addr*: RX-type address or address in register (2) - (12).

,CONSID=*console id* ,CONSNAME=*console name*	*console id*: RX-type address or register (2) - (12). *console name*: RX-type address or address in register (2) - (12).
,CMDFLAG=NOHCPY	
,TOKEN=*token*	*token*: RX-type address or register (2) - (12).
,UTOKEN=*utoken addr*	*utoken addr*: RX-type address or address in register (2) - (12).
,CART=*cart*	*cart*: RX-type address or address in register (2) - (12).
,MF=(E, *list addr*)	*list addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

TEXT=*text addr*
Specifies the required input field that contains the address of a command area. If a register is used, it should contain the address of the command area. The first 2 bytes of this command area contain the length of the command. The command text immediately follows this 2-byte area, and can be up to 126 characters. The command must be in storage addressable by the caller at the time the caller issues MGCRE.

Operator commands may contain the following characters:

- A to Z
- 0 to 9
- ' # $ & ( ) * + , – . / ¢ < | ! ; ¬ % _ > ? : @ " =

The system translates characters that are not valid into null characters (X'00').

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

,CONSID=*console id*
,CONSNAME=*console name*
CONSID specifies the required input field that contains the 4-byte ID of the console that issued the command specified in the TEXT parameter. If a register is used, it should contain the 4-byte console ID. If you specify CONSID, do not specify CONSNAME.

CONSNAME specifies the required input field that contains the console name. The console name is a 2- to 8-byte character string. If a register is used, it should contain the address of an 8-byte field containing the console name. This name identifies the console that issued the command specified in the TEXT parameter. The console name is left-justified and padded with blanks. If you specify CONSNAME, do not specify CONSID.

You must specify either CONSID or CONSNAME. Use the DISPLAY CONSOLES command to obtain these values.

Note: When you specify a console ID of X'00000000' on the CONSID parameter, the issuer receives MASTER command authority. Entries in the hardcopy log for the command have the name INTERNAL associated with them.

,CMDFLAG=NOHCPY
Requests that no copy of the command appear in the hardcopy log.

Note: If you do not specify this option, the system logs the command in the hardcopy log.

,TOKEN=*token*

Specifies the optional input field that contains a 31-bit right-justified program token for the command specified in the TEXT parameter. If a register is used, it should contain a 31-bit right justified token. Any 4-byte value is valid as input. TOKEN is optional.

,UTOKEN=*utoken addr*

Specifies the optional input field that contains the address of a security token for the command identified in the TEXT parameter. If a register is used, it should contain the address of a data area for the UTOKEN. You can obtain the UTOKEN value by using the RACROUTE REQUEST=TOKENXTR, RACROUTE REQUEST=VERIFYX, or RACROUTE REQUEST=TOKENBLD macros. See *OS/390 SecureWay Security Server External Security Interface (RACROUTE) Macro Reference* for more information on the RACROUTE macros. Command processing passes the UTOKEN to SAF (Security Authorization Facility) to validate the authority of the issuer. The UTOKEN should be that of the user on whose behalf the command is issued. UTOKEN is an optional parameter; if it is omitted, the address space's UTOKEN is used.

,CART=*cart*

Specifies the optional input field that contains the address of the 8-byte field that contains a command and response token. If a register is used, it should contain the address of a data area containing the command and response token. Your installation can use any value as a CART. The program that issues the command can tag each command with this token, which associates the command with its response. CART is an optional parameter.

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

Specifies the execute form of MGCRE. This form generates the code to store the parameters into the parameter list and execute the MGCRE macro.

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

ABEND Codes

MGCRE might abnormally terminate with abend code X'D22'. See *OS/390 MVS System Codes* for an explanation and programmer response for this code.

Return and Reason Codes

Register 15 contains one of the following hexadecimal return codes as the result of a START command. No return codes result from any other commands.

Figure 20. Return Codes for the START Command

Return Code	Meaning and Action
00	**Meaning**: The START command processed successfully. Register 0 contains the right-justified ASID of the started address space. **Action**: None.
08	**Meaning**: Environmental error. The START command failed for one of the following reasons: • The START command specified a console that is not authorized for entering the command • The system did not allow the address space to be created at this time due to a heavy system workload • There is not enough storage available to schedule the command • The system tried to obtain more address spaces than the maximum number supported. **Action**: Check to see if the START command specified a console that is not authorized for entering the command, and correct the situation if necessary. Next, retry the request. If the problem persists, record the return code and supply it to the appropriate IBM support personnel.

Example

Create the list form of MGCRE, modify it using the execute form of MGCRE, and issue a display consoles command associated with a console named CON4.

```
DOMTST   CSECT
R2       EQU   2
         USING *,R12
         LA    R2,CMD                    R2 POINTS TO THE COMMAND AREA
         MGCRE MF=(E,LAREA),TEXT=(R2),CMDFLAG=(NOHCPY),CONSNAME=MYCON
CMD      DS    0CL6                      THE COMMAND AREA
CMDLEN   DC    XL2'4'                    LENGTH OF COMMAND
CMDCOMM  DC    CL4'D C '                 THE ACTUAL COMMAND
MYCON    DC    CL8'CON4    '             NAME OF ISSUING CONSOLE
LAREA    MGCRE MF=L                      LIST FORM OF MGCRE
         END
```

MIHQUERY — Retrieve MIH Time Interval

Description

Use the MIHQUERY macro to retrieve the current MIH time interval setting for a device.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must reside in the primary address space

Programming Requirements

Before issuing the MIHQUERY macro, you must pin the UCB you specify in the macro. The UCB identifies the device. Pinning the UCB ensures that a reconfiguration request does not delete or reuse the UCB between the time you determine the UCB address and the time the MIHQUERY service uses the address.

For more information about pinning, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

Restrictions

None.

Input Register Information

Before issuing the MIHQUERY macro, the caller must ensure that the following general purpose register (GPR) contains the specified information:

Register	Contents
13	Address of an 18-word save area that must reside in the primary address space.

Output Register Information

When control returns to the caller of the MIHQUERY macro, the GPRs contain:

Register	Contents
0	Reason code
1	Address of the MIHQUERY control parameters
2-13	Unchanged
14	Return address
15	Return code

When control returns to the caller of the MIHQUERY macro, 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

© Copyright IBM Corp. 1988, 2000

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

To ensure that the system can detect missing interrupts, do not issue this macro more than once per second. Issuing the macro more than once per second might also interfere with DISPLAY, SET IOS, and SETIOS commands.

Syntax

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

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

UCBPTR=*ucb addr*	*ucb addr*: RS-type address, or address in register (2) - (12).
,TIMEINT=*time interval*	*time interval*: RS-type address, or address in register (2) - (12).
,RETCODE=*return code*	*return code*: RS-type address, or address in register (2) - (12).
,RSNCODE=*reason code*	*reason code*: RS-type address, or address in register (2) - (12).

Parameters

The parameters are explained as follows:

UCBPTR=*ucb addr*
A required parameter that specifies the fullword containing the address of the UCB or a copy of the UCB for the device whose MIH time interval you are requesting. To determine the UCB address, use the UCBSCAN or UCBLOOK macro, described in *OS/390 MVS Programming: Authorized Assembler Services Guide*.

,TIMEINT=*time interval*
A required parameter that specifies the fullword output field where the MIHQUERY service is to place the hexadecimal value of the MIH time interval, reported in seconds.

,RETCODE=*return code*
An optional parameter that specifies the location where the system is to place the return code. The return code is also in register 15.

,RSNCODE=*reason code*
An optional parameter that specifies the location where the system is to place the reason code. The reason code is also in register 0.

ABEND Codes

Any errors related to state, key, and addressing requirements cause an abend X'2C6'. See *OS/390 MVS System Codes* for complete information about this abend and its associated reason codes. To help debug the problem, provide a recovery routine that records and/or dumps the needed data, including the address and contents of the parameter list.

Return and Reason Codes

When the MIHQUERY macro returns control to your program, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code.

Figure 21. Return and Reason Codes for the MIHQUERY Macro

Return Code	Reason Code	Meaning and Action
00	None	**Meaning**: MIHQUERY processing completed successfully. **Action**: None.
04	None	**Meaning**: The MIH interval for this device is zero. The system does not monitor missing interrupts for the device. **Action**: None.
08	01	**Meaning**: Environmental error. NIP processing is still in progress. **Action**: Retry.
0C	None	**Meaning**: System error. This return code is for IBM diagnostic purposes only. **Action**: Supply the following information to the appropriate IBM support personnel: • Return codes • Dump data • LOGREC data

Example

Obtain an address of a UCB from the DASD class and pass that address to the MIHQUERY service. MIHQUERY determines the MIH time interval for the device.

```
*********************************************************************
**  USE THE LIST AND EXECUTE FORM OF THE MIHQUERY SERVICE.  OBTAIN   **
**  AN ADDRESS OF A UCB FROM THE DASD CLASS AND PASS THAT ADDRESS TO **
**  THE MIHQUERY SERVICE.                                           **
**                                                                  **
**  ASSUMPTIONS:                                                    **
**    ENTRY STATE: SUPERVISOR                                       **
**    ENTRY KEY  : 7                                                **
**    ENTRY MODE : PRIMARY                                          **
**                                                                  **
**    REGISTER USAGE:                                               **
**                                                                  **
**       BASEREG: 9                                                 **
**       DYNAMIC AREA: 6 (SUBPOOL 2)                                **
**                                                                  **
*********************************************************************
IOTMW111 CSECT ,
IOTMW111 AMODE 31
IOTMW111 RMODE ANY
*********************************************************************
** ENTRY LINKAGE                                                   **
*********************************************************************
         BAKR  14,0
         LR    9,15
         USING IOTMW111,9
*********************************************************************
** OBTAIN STORAGE FOR THE SERVICE PARAMETER LISTS AND WORKAREAS.    **
** ESTABLISH ADDRESSABILITY TO ALL.                                 **
*********************************************************************
```

```
         L     0,SIZDATD           GETS THE DYNAMIC AREA SIZE INTO
*                                  REGISTER ZERO FOR GETMAIN
         GETMAIN RU,LV=(0),SP=2    GETS THE DYNAMIC AREA FROM SUBPOOL 2
         LR    6,1                 GETS ADDRESS OF DYNAMIC AREA FROM
*                                  THE RETURNED ADDRESS OF THE GETMAIN
         USING MYDYNMIC,6          GETS ADDRESSABILITY TO THE DYNAMIC
*                                  AREA.
*********************************************************************
** SCANS FOR THE FIRST DEVICE IN THE DASD DEVICE CLASS.             **
** NOTE THAT THERE IS NO NEED TO PIN A COPY OF THE UCB FOR THE SCAN  **
** BUT A PIN IS REQUIRED FOR THE MIHQUERY SO IT IS DONE IN THE SCAN  **
** TO SAVE A SERVICE CALL.                                          **
*********************************************************************
         UCBSCAN ADDRESS,WORKAREA=UCBWORK,UCBPTR=MYUCBPTR,             X
               PIN,TEXT=MYPINTXT,PTOKEN=MYPTOKEN,                      X
               DYNAMIC=YES,RANGE=ALL,DEVCLASS=DASD,LINKAGE=SYSTEM,     X
               MF=(E,UCBAREA,COMPLETE)  GETS THE FIRST DASD DEVICE
*********************************************************************
*********  RETURN AND REASON CODES SHOULD BE CHECKED HERE  ************
*********************************************************************
*********************************************************************
**                  OBTAIN THE MIH TIME INTERVAL                   **
*********************************************************************
         MIHQUERY UCBPTR=MYUCBPTR,TIMEINT=TIMEINTERVAL,                X
               MF=(E,MIHAREA,COMPLETE)  QUERIES THE MIH INTERVAL FOR
*                                  THE DASD DEVICE.
*********************************************************************
*********  DO SOMETHING WITH THE RETURNED VALUE.    ******************
*********************************************************************
*********************************************************************
** UNPINS THE UCB.                                                  **
*********************************************************************
         UCBPIN UNPIN,PTOKEN=MYPTOKEN,LINKAGE=SYSTEM,                  X
               MF=(E,PINAREA,COMPLETE)
*********************************************************************
** RETURNS TO THE CALLER.                                           **
*********************************************************************
         PR                        RETURN TO CALLER
MYPINTXT DC    CL58'THIS SHOULD BE MEANINGFUL INFORMATION'
         LTORG
MYDYNMIC DSECT                     MY DYNAMIC AREA
*********************************************************************
*                                                                   **
* MIHQUERY LIST FORM                                                **
*                                                                   **
*********************************************************************
         MIHQUERY MF=(L,MIHAREA)
*********************************************************************
*                                                                   **
* UCBSCAN LIST FORM                                                 **
*                                                                   **
*********************************************************************
         UCBSCAN MF=(L,UCBAREA)
UCBWORK  DS    CL100               100 BYTE WORK AREA FOR UCBSCAN.
*********************************************************************
*                                                                   **
* UCBPIN LIST FORM                                                  **
*                                                                   **
*********************************************************************
         UCBPIN MF=(L,PINAREA)
MYPTOKEN DS    CL8                 PIN TOKEN RETURNED BY THE UCBSCAN
*                                  SERVICE.
TIMEINTERVAL DS CL4                MIH TIME INTERVAL RETURNED BY THE
*                                  MIHQUERY SERVICE.
```

```
MYUCBPTR DS    CL4                CONTAINS THE ADDRESS OF THE UCB
*                                 RETURNED BY THE SCAN SERVICE AND
*                                 FOR WHICH THE MIHQUERY IS DONE.
ENDDATD  DS    0D                 GETS ON AN 8 BYTE BOUNDARY FOR
*                                 GETMAIN
DYNSIZE  EQU   (ENDDATD-MYDYNMIC) TOTAL SIZE OF THE DYNAMIC AREA
IOTMW111 CSECT ,
         DS    0F
SIZDATD  DS    0A                 SETS THE SIZE IN THE MODULE
         DC    AL1(0)
         DC    AL3(DYNSIZE)
         END   IOTMW111
```

MIHQUERY—List Form

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

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

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 parameter is explained as follows:

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

MIHQUERY—Execute Form

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

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

UCBPTR=*ucb addr*	*ucb addr*: RS-type address, or address in register (2) - (12).
,TIMEINT=*time interval*	*time interval*: RS-type address, or address in register (2) - (12).
,RETCODE=*rc*	*rc*: RS-type address, or address in register (2) - (12).
,RSNCODE=*reason code*	*reason code*: RS-type address, or address in register (2) - (12).
,MF=(E,*list addr*) ,MF=(E,*list addr*,COMPLETE)	*list addr*: RS-type address, or address in register (2) - (12). **Default**: COMPLETE

Parameters

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

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

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

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that are not specified.

MODESET — Change System Status

Description

The MODESET macro is used to change system status by altering the PSW key and/or PSW problem state indicator. The MODESET macro has two forms: the form that generates inline code and the form that generates an SVC.

Inline Code Generation

Environment

The requirements for the caller are:

Minimum authorization:	Problem state, with any PSW key; also, the program must reside in an APF–authorized library and be link–edited with authorization code AC=1.
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or secondary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	None.

Programming Requirements

The caller must include the IKJTCB and IHARB mapping macros. If EXTKEY= TCB, RBT1, or RBT234 is specified, the home address space must be the currently addressable address space.

Restrictions

None.

Input Register Information

Before issuing the MODESET 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

All GPRs are unchanged except GPR 1 and 15, and those specified on the KEYADDR, KEYREG, SAVEKEY, and WORKREG keywords.

Register	Contents
0	Used as a work register by the system when specified on the WORKREG parameter; otherwise, unchanged.
1	Used as a work register by the system, even if you specify register 1 on the WORKREG or KEYREG parameter.
2	Used as a work register by the system when specified on the WORKREG, KEYREG, or KEYADDR parameters; otherwise, unchanged.
3-14	Used as work registers by the system when specified on the WORKREG, KEYREG, or KEYADDR parameters; otherwise, unchanged.
15	Used as a work register by the system.

When control returns to the caller, the ARs contain:

© Copyright IBM Corp. 1988, 2000

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 MODESET macro that generates inline code is written as follows:

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

EXTKEY=*key*	*key*: One of the following: 1. ZERO 2. TCB 3. RBT1 4. RBT234 5. KEY2 6. KEY3 7. KEY4 8. KEY7
KEYADDR=*new key addr* KEYREG=*new key reg*	*new key addr*: RX-type address or register (2). *new key reg*: Register 1-15 without parentheses; may be symbolic.
,SAVEKEY=*old key addr*	*old key addr*: RX-type address or register (2). **Notes:** 1. If KEYADDR=(2) is specified above, then SAVEKEY=(2) cannot be specified. 2. If WORKREG and SAVEKEY are specified with KEYREG, the KEYREG register should be different from the WORKREG register. Also, if SAVEKEY is specified with KEYREG, the KEYREG register should not be register 2.
,WORKREG=*work reg*	*work reg*: Decimal digits 0-15 without parentheses.

Notes:

1. WORKREG is required if the following are specified:

   EXTKEY=TCB
   EXTKEY=RBT234
   EXTKEY=RBT1

2. The WORKREG parameter should be register 1-15 if one of these four parameters is specified because WORKREG is used as a base register on the SPKA instruction. WORKREG=0 sets the PSW key to zero.

,RELATED=*value* — *value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

EXTKEY=*key*
Specifies the key, or the address of the key, to be set in the current PSW.

ZERO	Set a key of zero.
TCB	Set the key of the active TCB.
RBT1	Set the key of the active RB of type 1 SVC routine issuing MODESET.
RBT234	Set the key of the active RB preceding SVRB of type 2, 3, or 4 SVC routine issuing MODESET.
KEY2	Set a key of 2.
KEY3	Set a key of 3.
KEY4	Set a key of 4.
KEY7	Set a key of 7.

KEYADDR=*new key addr*
Specifies a location 1 byte in length which contains the key in bit positions 0-3. If register (2) is specified, the key is contained in bit positions 24-27 (bits 28-31 are ignored). This parameter permits a previously saved key to be restored.

KEYREG=*new key reg*
Specifies a register that contains a key value in bit positions 24-27.

,SAVEKEY=*old key addr*
Specifies a location 1 byte in length where the current PSW key is to be saved, in bit positions 0-3. If register (2) is specified, the key is left in register 2.

,WORKREG=*work reg*
Specifies the register into which the contents of register 2 are to be saved while performing the SAVEKEY function, or the working register to be used by the EXTKEY or KEYADDR function. If WORKREG=2 is specified, no register saving takes place.

,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

The MODESET macro might abnormally terminate with abend code X'0C2'. See *OS/390 MVS System Codes* for an explanation and programmer response.

Return and Reason Codes

None.

Example 1

Save the current PSW key, and change the key to that of the active TCB.

```
MODESET EXTKEY=TCB,SAVEKEY=KEYSAVE,WORKREG=1
```

Example 2

Save the current key at location KEY and set the key to the value contained in bits 24-27 of register 3.

```
MODESET KEYREG=REG3,SAVEKEY=KEY,WORKREG=4
```

SVC Generation

Environment

The requirements for the caller are:

Minimum authorization:	One of the following: - Supervisor state - PSW key 0 - 7 - APF authorization
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks held
Control Parameters:	Must be in primary address space

Programming Requirements

None.

Restrictions

None.

Input Register Information

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

Output Register Information

When control returns to the caller, the GPRs contain:

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

When control returns to the caller, the ARs contain:

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

2-13	Unchanged
14-15	Used as work registers by the system

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

Performance Implications

None.

Syntax

The standard form of the MODESET macro that generates an SVC is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede MODESET.
MODESET	
␣	One or more blanks must follow MODESET.
KEY=ZERO KEY=NZERO	**Note:** KEY is required if MODE is not specified.
,MODE=PROB ,MODE=SUP	**Note:** MODE is required if KEY is not specified.
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

KEY=ZERO
KEY=NZERO
Specifies that the PSW key (bits 8-11) is to be either set to zero (ZERO) or set to the value in the caller's TCB (NZERO).

,MODE=PROB
,MODE=SUP
Specifies that the PSW problem state indicator (bit 15) is to be either turned on (PROB) or turned off (SUP). If the MODESET operation completes with a problem state PSW, only the key specified by the problem state PSW will be authorized.

,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

The MODESET macro might abnormally terminate with abend code X'16B'. See *OS/390 MVS System Codes* for an explanation and programmer response.

Return and Reason Codes

When the MODESET macro returns control to your program, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code.

Figure 22. Return and Reason Codes for the MODESET Macro

Return Code	Reason Code	Meaning and Action
00	00	**Meaning**: The operation was successful. **Action**: None.

Example

Change to supervisor mode and key zero.

```
MODESET KEY=ZERO,MODE=SUP
```

MODESET—List Form

Syntax

The list form of the MODESET macro that generates an SVC is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede MODESET.
MODESET	
␣	One or more blanks must follow MODESET.
KEY=ZERO KEY=NZERO	**Note:** KEY is required if MODE is not specified.
,MODE=PROB ,MODE=SUP	**Note:** MODE is required if KEY is not specified.
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

Parameters

The parameters are explained under the standard form of the MODESET macro that generates an SVC, with the following exception:

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

MODESET—Execute Form

Syntax

The execute form of the MODESET macro that generates an SVC is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede MODESET.
MODESET	
␣	One or more blanks must follow MODESET.
RELATED=*value,*	*value*: Any valid macro keyword specification.
,MF=(E,*list addr*)	*list addr*: RX-type address, or register (1).

Parameters

The parameters are explained under the standard form of the MODESET macro that generates an SVC, with the following exception:

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

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

NIL — Provide a Lock Via an AND IMMEDIATE (NI) Instruction

Description

The NIL macro is used to provide a lock on a byte of storage on which an AND immediate (NI) instruction is to be executed. Because the byte of storage exists in a multiprocessing environment, the possibility exists that the byte might be changed by another processor at the same time. Storage modification during NIL processing is accomplished by using the compare and swap (CS) instruction.

For details on the AND immediate and compare and swap instructions, see *Principles of Operation*.

Environment

These are the requirements for the caller:

Minimum authorization:	Problem or supervisor state, any 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:	None
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 NIL 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 are unchanged except for the three work registers that are used by the system. If WREGS is not specified, these will be registers 0-2.

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.

© Copyright IBM Corp. 1988, 2000

Syntax

The NIL macro is written as follows:

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

byte addr	*byte addr*: RX-type address.
,mask	*mask*: Symbol or self-defining term.
,REF=*stor addr*	*stor addr*: RX-type address.
,WREGS=*(reg1,reg2,reg3)* ,WREGS=*(reg1,reg2)* ,WREGS=*(reg1,,reg3)* ,WREGS=*(,reg2,reg3)* ,WREGS=*(reg1)* ,WREGS=*(,reg2)* ,WREGS=*(,,reg3)*	*reg1*: Symbol or decimal digits 0-15. *reg2*: Symbol or decimal digits 1-15. *reg3*: Symbol or decimal digits 0-15. **Default** for *reg1*: 0 **Default** for *reg2*: 1 **Default** for *reg3*: 2

Parameters

The parameters are explained as follows:

byte addr
Specifies the address of the byte to which the AND function is to be applied.

,*mask*
Specifies the value to be ANDed to the byte at the address specified above.

,REF=*stor addr*
Specifies the address of a storage location on a fullword boundary. This address provides the means by which the compare and swap instruction may be executed. The address must be less than or equal to the byte address specified above, and the difference between the addresses must be less than 4095. The two addresses must be addressable via the same base register.

,WREGS=*(reg1,reg2,reg3)*
,WREGS=*(reg1,reg2)*
,WREGS=*(reg1,,reg3)*
,WREGS=*(,reg2,reg3)*
,WREGS=*(reg1)*
,WREGS=*(,reg2)*
,WREGS=*(,,reg3)*
Specifies the work registers to be used to perform the compare and swap instruction. *reg1* is used to contain the “old” byte; *reg2* is used to contain the “updated” byte; and *reg3* is used to contain the mask.

ABEND Codes

None.

Return and Reason Codes

None.

Example

Turn off bit TNVLXMET in byte TNVLCS1. The reference field, TNVLFW3, specifies the word being updated.

```
NIL TNVLCS1,X'FF'-TNVLXMET,REF=TNVLFW3
```

NMLDEF — Customizing the Nucleus

Description

A set of tables, called nucleus module lists (NMLs), are used to identify the members in SYS1.NUCLEUS that are to be loaded into the DAT-on nucleus region. NMLs can be installed as part of an IBM product, a vendor product, or a customer user modification. Each NML contains a list of the SYS1.NUCLEUS members that are part of the same product or user modification. The NMLs themselves are load modules that also reside in SYS1.NUCLEUS.

The NML must have a module name in the form of IEANC*nnn* where *nnn* is a 3-digit number from 001 through 256.

Use the NMLDEF macro to generate an NML statement (at the end of the macro expansion). See *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information on using the NMLDEF macro.

Existing nucleus-resident entry points cannot be replaced or overridden using an NML. To find out how to perform these functions, refer to information about customizing the nucleus in the *OS/390 MVS Programming: Authorized Assembler Services Guide*.

You can use a NUCLSTxx member of SYS1.PARMLIB instead of NMLDEF to specify modules to be loaded into the nucleus. For more information on NUCLSTxx, especially its possible advantages over NMLDEF, see *OS/390 MVS Initialization and Tuning Reference*.

Environment

The requirements for the caller are:

Minimum authorization:	Problem state or supervisor state, and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	No requirement
AMODE:	24- or 31-bit
ASC mode:	No requirement
Interrupt status:	No requirement
Locks:	No requirement
Control parameters:	None.

Note: This macro does not generate any executable code.

Programming Requirements

None.

Restrictions

None.

Register Information

None.

© Copyright IBM Corp. 1988, 2000

Performance Implications

None.

Syntax

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

name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede NMLDEF.
NMLDEF	
ƀ	One or more blanks must follow NMLDEF.
NUCL=*nucid*	*nucid*: One to eight characters in length.

Parameters

The parameter is explained as follows:

NUCL=*nucid*
Identifies the name of one or more SYS1.NUCLEUS members that are to be loaded into the nucleus region. At least one nucleus identifier must be specified on the NMLDEF macro. If you specify more than one nucleus identifier, enclose the list in parentheses and use commas to separate the identifiers.

ABEND Codes

None.

Return and Reason Codes

None.

Example 1

Specify the macro as follows to name an NML as IEANC002 and identify the members named ABC00122 and ABC00123 in SYS1.NUCLEUS that are to be loaded into the nucleus region.

```
IEANC002  NMLDEF  NUCL=(ABC00122,ABC00123)
```

Example 2

To install a user nucleus-resident routine or a release of a product, you can use SMP/E, or code your own JCL.

The following example JCL creates an NML with the CSECT name of IEANC001 containing the module name of DXXTEST.

```
//FORDON    JOB   MSGLEVEL=(1,1)
//NMLDEF    EXEC  ASMHCL
//ASM.SYSIN    DD    *
IEANC001  NMLDEF NUCL=DXXTEST
//LKED.SYSLMOD DD    DSN=SYS1.NUCLEUS,VOL=SER=DCH352,UNIT=3380,DISP=OLD
//LKED.SYSIN   DD    *
  NAME IEANC001
/*
```

Example 3

To install a user nucleus-resident routine or a release of a product, you can use SMP/E, or code your own JCL.

The following example JCL creates NMLs with the CSECT name of IEANC002 containing module names of ABC00001-ABC00010.

```
//FORDON    JOB   MSGLEVEL=(1,1)
//NMLDEF    EXEC  ASMHCL
//ASM.SYSIN    DD    *
IEANC002  NMLDEF NUCL=(ABC00001,ABC00002,ABC00003,ABC00004,ABC00005,    X
               ABC00006,ABC00007,ABC00008,ABC00009,ABC00010)
//LKED.SYSLMOD DD    DSN=SYS1.NUCLEUS,VOL=SER=DCH352,UNIT=3380,DISP=OLD
//LKED.SYSIN   DD    *
  NAME IEANC002
/*
```

NUCLKUP — Nucleus Map Lookup Service

Description

The NUCLKUP macro provides lookup functions by name and by address. Use the macro to get:

- The name and address of a nucleus CSECT if you have an address within the CSECT
- The address and AMODE of a nucleus CSECT or ENTRY if you have its name

You would use the NUCLKUP macro to:

- Check an installation program once your installation has put it into the nucleus
- Diagnose a problem of a program residing in the nucleus

Environment

The requirements for the caller are:

Minimum authorization:	Problem state 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
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space

Programming Requirements

The caller must include the CVT mapping macro in the program that issues the NUCLKUP macro.

Restrictions

None.

Input Register Information

Before issuing the NUCLKUP macro, the caller must ensure that the following general purpose register (GPR) contains the specified information:

Register	Contents
13	The address of a standard 18-word save area

Output Register Information

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

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

Register	Contents
0	For a BYNAME request, the address and AMODE of the CSECT or ENTRY; for a BYADDR request, the 31-bit address of the CSECT.

© Copyright IBM Corp. 1988, 2000

1	For a BYNAME request, the high-order byte is zero and the low-order three bytes contain the length from the entry point to the end of the CSECT; for a BYADDR request, register 1 contains the address of the 8-byte area in which the CSECT name is returned.
2-14	Unchanged, except if you specify BYNAME,NAME=*name id*,ADDR=*addr*, and *addr* is a register. In this case, the register you specify will contain the address of the CSECT or ENTRY.
15	Return code.

Performance Implications

None.

Syntax

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

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

BYNAME,NAME=*name id*	*name id*: 8-byte literal (enclosed in apostrophes), or the address of the 8-byte literal which can be either an RX-type address, or register (1) - (12).
BYADDR,NAME=*name loc*	*name loc*: RX-type address or register (1) - (12).
,ADDR=*addr*	*addr*: RX-type address, or register (0) or (2) - (12).

Parameters

The parameters are explained as follows:

BYNAME
BYADDR
Specifies the function to be performed. If BYNAME is specified, the user supplies on NAME=*name id* the name of a CSECT or ENTRY and receives on ADDR the address and AMODE of that CSECT or ENTRY. If BYADDR is specified, the user supplies on ADDR an address within a CSECT and receives on NAME=*name id* the name and address of the CSECT.

,NAME=*name id*
,NAME=*name loc*
Specifies the name or the location of the name of the CSECT depending on the option requested. If the user specifies BYNAME, *name id* contains the 8-character name to be searched for or the address of that name. If the user specifies BYADDR, *name loc* will contain the address of the 8-byte area in which the CSECT name is to be returned.

,ADDR=*addr*
Contains the address to be searched for if BYADDR is specified; contains the address of the CSECT or ENTRY that is returned if BYNAME is specified.

The NUCLKUP service routine sets bit 0 of the word containing the address returned on a BYNAME request to indicate the AMODE of the CSECT. For example, if the requestor's AMODE is 31-bit and the AMODE of the CSECT is ANY, the NUCLKUP service routine sets bit 0 to 1. The setting of bit 0 is summarized in the following table:

Requestor's AMODE	AMODE of CSECT		
	24	31	ANY
24	0	1	0
31	0	1	1

ABEND Codes

None.

Return Codes

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

Figure 23. Return Codes for the NUCLKUP Macro

Return Code	Meaning and Action
00	**Meaning**: The request was satisfied. **Action**: None.
04	**Meaning**: The request was not satisfied. For a BYNAME request, the name was not found, and the location containing the address was set to zero. For a BYADDR request, the address was not found in the nucleus, and the location containing the name was set to zero. **Action**: None required. However, you might take some action based upon your application.
08	**Meaning**: Program error. The request was not satisfied because the type of request was not specified correctly. The locations containing the name and address were set to zero. **Action**: Ensure that the name id value is supplied for BYNAME requests, and the *addr* value is provided on BYADDR requests.

Example 1

Place the address and AMODE of entry point IEAVESTU in register 0.

```
NUCLKUP BYNAME,NAME='IEAVESTU',ADDR=(0)
```

Example 2

Look up the address and amode of the entry point mane in location STRING and return it at location RETLOC.

```
        NOCLKUP BYNAME,NAME=STRING,ADDR=RETLOC
        .
        .
STRING  DS      CL8
RETLOC  DS      F
```

Example 3

Return the CSECT name and address of the nucleus routine in which the address at location INADDR falls. Return the name at location EPLOC1 and the address at INADDR.

```
        NUCLKUP BYADDR,NAME=EPLOC,ADDR=INADDR
        .
        .
EPLOC   DS      CL8
INADDR  DS      F
```

OIL — Provide a Lock Via an OR IMMEDIATE (OI) Instruction

Description

The OIL macro is used to provide a lock on a byte of storage on which an or immediate (OI) instruction is to be executed. Because the byte of storage exists in a multiprocessing environment, the possibility exists that the byte might be changed by another processor at the same time. Storage modification during OIL processing is accomplished by using the compare and swap (CS) instruction.

For details on the or immediate and compare and swap instructions, see *Principles of Operation*.

Environment

These are the requirements for the caller:

Minimum authorization:	Problem or supervisor state, any 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:	None
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 OIL 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 are unchanged except for the three work registers that are used by the system. If WREGS is not specified, these will be registers 0-2.

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.

© Copyright IBM Corp. 1988, 2000

Syntax

The OIL macro is written as follows:

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

byte addr	*byte addr*: RX-type address.
,mask	*mask*: Symbol or self-defining term.
,REF=*stor addr*	*stor addr*: RX-type address.
,WREGS=*(reg1,reg2,reg3)* ,WREGS=*(reg1,reg2)* ,WREGS=*(reg1,,reg3)* ,WREGS=*(,reg2,reg3)* ,WREGS=*(reg1)* ,WREGS=*(,reg2)* ,WREGS=*(,,reg3)*	*reg1*: Symbol or decimal digits 0-15. *reg2*: Symbol or decimal digits 0-15. *reg3*: Symbol or decimal digits 0-15. **Default**: for *reg1*: 0 **Default**: for *reg2*: 1 **Default**: for *reg3*: 2

Parameters

The parameters are explained as follows:

byte addr
Specifies the address of the byte to which the OR function is to be applied.

,mask
Specifies the value to be ORed to the byte at the address specified above.

,REF=*stor addr*
Specifies the address of a storage location on a fullword boundary. This address provides the means by which the compare and swap instruction may be executed. The address must be less than or equal to the byte address specified above, and the difference between the addresses must be less than 4095. The two addresses must be addressable via the same base register.

,WREGS=*(reg1,reg2,reg3)*
,WREGS=*(reg1,reg2)*
,WREGS=*(reg1,,reg3)*
,WREGS=*(,reg2,reg3)*
,WREGS=*(reg1)*
,WREGS=*(,reg2)*
,WREGS=*(,,reg3)*
Specifies the work registers to be used to perform the compare and swap instruction. *reg1* is used to contain the “old” byte; *reg2* is used to contain the “updated” byte; and *reg3* is used to contain the mask.

ABEND Codes

None.

Return and Reason Codes

None.

Example

Turn on bit TVNLXMET in byte TVNLCS1. The reference field TVNL specifies the area containing the word being updated.

```
OIL TVNLCS1,TVNLXMET,REF=TVNL
```

OUTADD — Create an Output Descriptor

Description

Use the OUTADD macro to create an output descriptor for a system output (sysout) data set. For information about using the OUTADD macro, see “Dynamic Output” in *OS/390 MVS Programming: Authorized Assembler Services Guide*.

The OUTADD macro has no standard form. Use the list form to generate a storage declaration for the input parameter list to dynamic output. Use the execute form to modify the parameter list and invoke dynamic output.

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

When you code the input data to dynamic output, all character data must be left-justified in a field of blanks. Any noncharacter input data must be right-justified in a field of binary zeroes. All input addresses or pointers must be coded as 31-bit addresses, even when you invoke dynamic output in a 24-bit addressing environment. If you use a 24-bit address, you must right-justify the address in a 4-byte field, and set the left-most byte to binary zeroes.

Restrictions

- You can use dynamic output in a JES2 environment or a JES3 4.2.1 or later environment.
- Output descriptors are deleted with the OUTDEL macro. On some systems, the system does not free the output descriptor's storage when you delete the output descriptor. Whether or not this storage is freed depends on the version of JES that is being used on your system. For more information, see “Dynamic Output” in *OS/390 MVS Programming: Authorized Assembler Services Guide*.
- When you use OUTADD to create output descriptors in a program that also uses checkpoint/restart, you must observe the restrictions that are described in *OS/390 DFSMS Checkpoint/Restart*.

Input Register Information

Before issuing the OUTADD 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, 2000

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0	Reason code
1	Key of the failing text unit, if available; otherwise, zero
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

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.

OUTADD—List Form

Syntax

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

name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede OUTADD.
OUTADD	
ƀ	One or more blanks must follow OUTADD.
MF=L	

The parameters of the list form, which are both required, are explained as follows:

name
: The list form defines the storage area to be used as the input parameter list to the OUTADD macro. *name* specifies the symbolic address of this storage.

MF=L
: Specifies the list form of the OUTADD macro.

Example

Use the list form of the OUTADD macro to generate the input parameter list that is to be used by the execute form of the OUTADD macro. Locate the parameter list at symbolic location, PARML.

```
PARML    OUTADD  MF=L
```

OUTADD—Execute Form

The execute form of the OUTADD macro modifies and executes the parameter list that was built with the list form of the OUTADD macro.

Syntax

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

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

NAME=*descriptor name addr* SYSNAME=*descriptor name addr*	*descriptor name addr*: Rx-type address or register (2)-(12).
,TEXTPTR=*tu pointer addr*	*tu pointer addr*: Rx-type address or register (2)-(12).
,ENQ=CONDITIONAL ,ENQ=UNCONDITIONAL	**Default**: ,ENQ=UNCONDITIONAL
,MF=(E,*list addr*)	*list addr*: Rx-type address or register (2)-(12).

Parameters

The parameters are explained as follows:

NAME=*descriptor name addr*

Specifies the address of the location where you place the name of the output descriptor. The name, which must be unique in the current job step, must be one to eight characters long. If less than eight characters, the name must be left justified in an eight byte field of blanks. The first character must be alphabetic or national (@, $, #), and the remaining characters can be alphanumeric or national.

NAME is mutually exclusive with SYSNAME. You must specify either NAME or SYSNAME.

Note: If you pass a name field of all binary zeros, the system generates a name and uses that name. In this case, however, the system does not return the name.

SYSNAME=*descriptor name addr*

Specifies the address of an 8-character field into which the system is to return an output descriptor name. SYSNAME requests the system to generate the output descriptor name. SYSNAME is mutually exclusive with NAME. You must code either SYSNAME or NAME.

,TEXTPTR=*tu pointer addr*

Specifies the address of the text unit pointer list that you create. It is a required parameter. The text unit pointer list references your text units; to invoke OUTADD, at least one text unit must exist.

Place the pointer to each of your text units into the text unit pointer list. The pointers in the text unit pointer list must be in contiguous storage. Each pointer must be four bytes long, be aligned on a fullword boundary, and point to a single text unit. Alternatively, if bits 1 - 31 of the text unit pointer are zero, dynamic output assumes no text unit is associated with the text unit pointer. You can use a text unit pointer with zeros in bits 1 - 31 to avoid having to rearrange a predefined text unit pointer list when you do not want to point to a particular text unit.

To enable the system to find the end of the pointer list, set the leftmost bit of the last pointer to 1. The system limits the number of text unit pointers to 1000. For a coded example of a text unit pointer list, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

,ENQ=CONDITIONAL
,ENQ=UNCONDITIONAL

Specifies whether the OUTADD create request is to wait on a system queue. When you invoke dynamic output to create an output descriptor, the system serializes the create request. Because the system processes the requests one at a time, each must wait on a system queue for all the previous requests to be processed. If you do not want your request to wait, code ENQ=CONDITIONAL. This causes the system to ignore your create request unless the queue is empty. You get return code 04 with reason code of X'405' when the system ignores your create request. To allow your request to go on the queue, code ENQ=UNCONDITIONAL. If you omit the ENQ parameter, your request always goes on the queue.

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

Specifies the execute form of the OUTADD macro.

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

ABEND Codes

OUTADD might abnormally terminate with abend codes X'76D' or X'054'. See *OS/390 MVS System Codes* for explanations and programmer responses.

Return and Reason Codes

When control returns to the caller, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code. Note that when you invoke dynamic output to create an output descriptor, the system creates the descriptor only if the return code is zero.

If the dynamic output request fails, you get a nonzero return code in GPR 15 and a reason code in GPR 0. If any text unit can be associated with the failure, the key of the failing text unit is in the rightmost two bytes of GPR 1. GPR 1 contains zero if the failure does not implicate a particular text unit, or if a text unit caused the failure but the system cannot determine what text unit is responsible.

When you are using a reason code to debug your program, it is sometimes advisable to look beyond the immediate explanation of the reason code. For example, even if the reason code indicates a bad text unit, the text units you coded might be perfectly correct. The pointers to the text units, however, might be incorrectly coded, or one of the pointers in the pointer list might be bad.

For programmers who want them, symbolic names for reason codes are available in the IEFDORC mapping macro. (See *OS/390 MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.) Reason codes are four bytes long.

Figure 24. Return and Reason Codes for the OUTADD Macro		
Return Code	**Reason Code**	**Meaning and Action**
00	000	**Meaning**: The dynamic output request is successful. **Action**: None.
04	400-408	**Meaning**: The dynamic output request failed because of an error in dynamic output or elsewhere in the system.
08	380-391 500-504	**Meaning**: The dynamic output request failed. There is an error in an operating system program. **Action**: Contact the system programmer.
08	000 6000-7FFF	**Meaning**: The dynamic output request failed. The request was denied by your installation. These reason codes are defined by your installation.
0C	300-314	**Meaning**: The dynamic output request failed. The program that invoked dynamic output has built the text units incorrectly or, less likely, the installation exit routine has built the text units incorrectly.
0C	380-394	**Meaning**: The dynamic output request failed because the parameter list is incorrectly initialized.
0C	409	**Meaning**: The dynamic output request failed. The caller's text unit pointer list was altered by another task during dynamic output processing.
10	700-702	**Action**: An abend occurred in the system. **Action**: Consult the system programmer.

Reason Codes for Return Code 04

The table below documents the reason codes that can occur when the OUTADD macro returns with a return code of 04 (in GPR15). The symbol for return code 04 is DOENVERR in macro IEFDORC. The “symbol” field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro IEFDORC.

Figure 25 (Page 1 of 2). Reason Codes for Return Code 04	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
400 (1024)	**Symbol**: DORCGET1 **Meaning**: Environmental error. A GETMAIN request failed during processing of the dynamic output request. **Action**: Attempt to allow a larger region for the application and retry the dynamic output request.
401 (1025)	**Symbol**: DORCEXST **Meaning**: Environmental error or program error. The dynamic output request specified an output descriptor that already exists. **Action**: Select an alternate name for the output descriptor and retry the dynamic output request.
404 (1028)	**Symbol**: DORCESTA **Meaning**: Program or system error. During dynamic output processing, the system was unable to establish a recovery environment. **Action**: Determine if your program is doing something that prevents a recovery environment from being established. The most likely reason for this error is that your program is causing all available storage to be used. If the problem cannot be attributed to your program, record the return and reason code, and optionally make a copy of your application. Give this information to your system programmer to supply to the appropriate IBM support personnel.
405 (1029)	**Symbol**: DORCNENQ **Meaning**: Environmental error. The ENQ resource is not available at this time. This reason code can be issued only for conditional ENQ requests. **Action**: Run your application at another time, when the system queue is empty. Alternatively, you can specify ENQ=UNCONDITIONAL (the default), and the request will always go onto the queue.

Figure 25 (Page 2 of 2). Reason Codes for Return Code 04	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
406 (1030)	**Symbol**: DORCNONM **Meaning**: Environmental error. No more system-generated names can be created; the maximum number allowed are in use. **Action**: If your application can delay processing the dynamic output request, it is possible that an OUTDEL request will be issued. This OUTDEL request might allow additional system-generated names to be created. You can also consider segmenting the work of the application so that a smaller number of system-generated names are in use at any point in time.
407 (1031)	**Symbol**: DORCGET2 **Meaning**: Environmental or system error. A GETMAIN request failed during processing of the dynamic output request. Possible causes include: • You created a large number of output descriptors and have not deleted them • Another program in the region is using up a lot of storage. **Action**: Delete any output descriptors that should have been deleted. Attempt to allow a larger region for the application and retry the dynamic output request.
408 (1032)	**Symbol**: DORCALTT **Meaning**: Environmental or program error. A text unit was found to have been altered by another task running in the system. **Action**: Review the design of the application and determine if additional tasks running concurrently with this dynamic output request could be altering shared storage (the text unit). If a change can be made to eliminate the storage alteration by another task, retry the dynamic output request.

Reason Codes for Return Code 08

The table below documents the reason codes that can occur when the OUTADD macro returns with a return code of 08 (in GPR15). The symbol for return code 08 is DOREQDNY in macro IEFDORC. The "symbol" field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro IEFDORC.

Figure 26 (Page 1 of 4). Reason Codes for Return Code 08	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
0 (0)	**Symbol**: (none defined) **Meaning**: Program error. The dynamic output request was denied by your installation. **Action**: The meaning of this reason code is defined by your installation. Your installation should be able to make recommendations for altering the dynamic output request to conform to installation standards.
380 (896)	**Symbol**: DORCLNIV **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Dynamic output processing detected an incorrect parameter list length. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNLEN field was incorrectly set and correct the problem.
381 (897)	**Symbol**: DORCNZF1 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the unused bits in the DOCNFNC1 field are not zero and correct the problem.

Figure 26 (Page 2 of 4). Reason Codes for Return Code 08

Reason Code Hexadecimal (Decimal)	Meaning and Action
382 (898)	**Symbol**: DORCNZF2 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the unused bits in the DOCNFNC2 field are not zero and correct the problem.
383 (899)	**Symbol**: DORCNZR1 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNRSV1 reserved field is not zero and correct the problem.
384 (900)	**Symbol**: DORCNZR2 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNRSV2 reserved field is not zero and correct the problem.
385 (901)	**Symbol**: DORCIVID **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A portion of the dynamic output parameter list was improperly set. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNID field was not properly initialized and correct the problem.
386 (902)	**Symbol**: DORCIVVR **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. The DOCNVERS field contains an incorrect parameter list length. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNVERS field was not properly initialized and correct the problem.
387 (903)	**Symbol**: DORCNOFN **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A portion of the dynamic output parameter list was improperly set. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Either the DOCNNEW or the DOCNDEL bit must be set in the DOCNFNC1 field. Determine why no function bit (create or delete) was set and correct the problem.

Figure 26 (Page 3 of 4). Reason Codes for Return Code 08

Reason Code Hexadecimal (Decimal)	Meaning and Action
388 (904)	**Symbol**: DORCIVFN **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A portion of the dynamic output parameter list was improperly set. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Only one function bit (either DOCNNEW or DOCNDEL) should be set in the DOCNFNC1 field. Determine why more than one function bit (create or delete) was set and correct the problem.
38B (907)	**Symbol**: DORCIVNM **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Dynamic output processing received an incorrect output descriptor name from the installation exit. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). The output descriptor name is in the DOCNNAME field. Check the output descriptor name returned to the dynamic output service. The first character must be an alphabetic (capitalized) character or a national character (#, @, or $). The remaining characters must be alphabetic (capitalized) characters, national characters (#, @, or $), or numbers. There can be no intervening blanks. Alternatively, the name can be all (binary) zeros, indicating that the system should generate a name for this request.
38F (911)	**Symbol**: DORCIVTU **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A portion of the dynamic output parameter list was improperly set. The installation exit maps the dynamic output parameter list (IEFDOCNP). DOCNTXTP must point to a text unit pointer list. Determine why DOCNTXTP was set to zero and correct the problem.
391 (913)	**Symbol**: DORCNZR0 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNRSV0 reserved field is not zero and correct the problem.
500 (1280)	**Symbol**: DORCINST **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is 8, but the reason code is not within the allowable range defined for the dynamic output installation exit. Determine why the installation exit is returning a reason code that is outside the allowable range and correct the error.
501 (1281)	**Symbol**: DORCINRC **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is zero, but the reason code is not zero. Determine why the installation exit is returning a nonzero reason code but a zero return code for the dynamic output request and correct the problem.

Figure 26 (Page 4 of 4). Reason Codes for Return Code 08

Reason Code Hexadecimal (Decimal)	Meaning and Action
502 (1282)	**Symbol**: DORCINRT **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is not within the allowable range defined for the dynamic output installation exit. Determine why the installation exit is returning a return code that is outside the allowable range and correct the error.
503 (1283)	**Symbol**: DORCINKE **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is zero, but the installation exit has returned a nonzero value in register 1. When control is returned from the installation exit, register 1 contains the value of the key that is in error. Determine why the installation exit is returning a nonzero value in register 1 and a zero return code for the dynamic output request.
504 (1284)	**Symbol**: DORCZKEY **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The installation exit has modified the input from the user. It now has a zero text unit key which is rejected by later processing within dynamic output.
6000-7FFF (24576-32767)	**Symbol**: (none defined) **Meaning**: Program error. The dynamic output request was denied by your installation. **Action**: The meaning of this reason code is defined by your installation, and indicates the reason the dynamic output request was denied. Your installation should be able to make recommendations for altering the dynamic output request to conform to installation standards.

Reason Codes for Return Code 0C

OUTADD returns a return code of 0C for errors detected in the caller's parameters. The list form of the OUTADD macro generates a parameter list. When the caller invokes the execute form of the OUTADD macro, this parameter list is filled in with the parameters coded on the execute form of the OUTADD macro. The OUTADD service verifies this parameter list and text units pointed to from this parameter list.

The table below documents the reason codes that can occur when the OUTADD macro returns with a return code of 0C (in GPR15). The symbol for return code 0C is DOINVPRM in macro IEFDORC. The “symbol” field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro IEFDORC.

Note: Reason codes less than 380 (hexadecimal) can be returned if the dynamic output installation exit modifies the text units associated with the request, producing an error that is detected later by dynamic output processing.

Figure 27 (Page 1 of 5). Reason Codes for Return Code 0C

Reason Code Hexadecimal (Decimal)	Meaning and Action
300 (768)	**Symbol**: DORCIVCH **Meaning**: Program error. A selection specified for a value field in a text unit was not valid. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
301 (769)	**Symbol**: DORCGMAX **Meaning**: Program error. A numeric parameter exceeds the maximum allowable value. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
302 (770)	**Symbol**: DORCLMIN **Meaning**: Program error. A numeric parameter is less than the minimum allowable value. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
303 (771)	**Symbol**: DORCNNUM **Meaning**: Program error. No parameter was specified for the text unit key. **Action**: This is probably a logic error in the application program. Correct the problem and retry the dynamic output invocation.
306 (774)	**Symbol**: DORCNLLN **Meaning**: Program error. A level name in the value field of a text unit is too long. The maximum is 8 characters. For example, `ACMESYSTEM.USNA` is not valid because `ACMESYSTEM` contains more than 8 characters. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
307 (775)	**Symbol**: DORCNLNM **Meaning**: Program error. There are too many level names in the value field of a text unit. For example, `DAVE.ACME.SYSR` is wrong if only two levels are allowed. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
308 (776)	**Symbol**: DORCNFCH **Meaning**: Program error. The first character in the name of a level in the value field of a text unit is incorrect. For example, `4DAVE.ACME` or `DAVE.4ACME` is incorrect if numeric characters are not allowed as the first character of a level name. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
309 (777)	**Symbol**: DORCNOCH **Meaning**: Program error. In the value field of a text unit, a character (other than the first) in a level name is incorrect. For example, `GARY3.SINKHOLE` and `GARY.SI8NK` are incorrect if only the first character in a level name can be a numeric character. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
30A (778)	**Symbol**: DORCNLIV **Meaning**: Program error. In the value field of a text unit, the levels are incorrectly specified. The value field has two consecutive periods, or the first or last character of the value field is a period. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
30B (779)	**Symbol**: DORCIVNP **Meaning**: Program error. The number of parameters in the text unit is not valid. **Action**: The number of parameters in the text unit was indicated as being less than zero or greater than 1000. Modify the text unit so that the number of parameters is within the allowable range. Retry the dynamic output invocation.

Figure 27 (Page 2 of 5). Reason Codes for Return Code 0C

Reason Code Hexadecimal (Decimal)	Meaning and Action
30C (780)	**Symbol**: DORCIVLN **Meaning**: Program error. The length field in the text unit is not valid. **Action**: The length field in the text unit was indicated as being less than zero or greater than 254. Modify the text unit length so that it is within the allowable range. Retry the dynamic output invocation.
30D (781)	**Symbol**: DORCNKEY **Meaning**: Program error. The key in the text unit was not valid. **Action**: This is probably a logic error in the application program. Correct the problem and retry the dynamic output invocation.
30E (782)	**Symbol**: DORCDUPK **Meaning**: Program error. Two or more text units were specified with identical keys. **Action**: This is probably a logic error in the application program. Correct the problem and retry the dynamic output invocation.
30F (783)	**Symbol**: DORCIVKY **Meaning**: Program error. A key that was not valid was found in a text unit. **Action**: Determine why a key that was not valid was found in the text unit, correct the problem, and retry the dynamic output request.
310 (784)	**Symbol**: DORCNSLE **Meaning**: Program error. An incorrect subparameter has been specified for a text unit. The specified subparameter is not defined for this text unit. **Action**: This is probably a logic error in the application program. Correct the problem and retry the dynamic output invocation.
311 (785)	**Symbol**: DORCMTUP **Meaning**: Program error. The number of text unit pointers was found to be too large. **Action**: The number of text unit pointers must be less than 1000. Modify the text unit pointer list so that there are less than 1000 text unit pointers. Retry the dynamic output invocation.
312 (786)	**Symbol**: DORCIVTX **Meaning**: Program error. An incorrect text character was detected in the value field of the text unit. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
313 (787)	**Symbol**: DORCISEQ **Meaning**: Program error. A character sequence that was not valid was specified in the value field of the text unit. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
314 (788)	**Symbol**: DORCIBIT **Meaning**: Program error. During the specification of a text unit that defines a bitstring-type parameter, a bit was specified that is not allowed. **Action**: This is probably a logic error in the application program, or perhaps incorrect data was propagated by the application program to the dynamic output request. Correct the problem and retry the dynamic output invocation.
380 (896)	**Symbol**: DORCLNIV **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.

Figure 27 (Page 3 of 5). Reason Codes for Return Code 0C

Reason Code Hexadecimal (Decimal)	Meaning and Action
381 (897)	**Symbol**: DORCNZF1 **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
382 (898)	**Symbol**: DORCNZF2 **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
383 (899)	**Symbol**: DORCNZR1 **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
384 (900)	**Symbol**: DORCNZR2 **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
385 (901)	**Symbol**: DORCIVID **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
386 (902)	**Symbol**: DORCIVVR **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
387 (903)	**Symbol**: DORCNOFN **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.

Figure 27 (Page 4 of 5). Reason Codes for Return Code 0C

Reason Code Hexadecimal (Decimal)	Meaning and Action
388 (904)	**Symbol**: DORCIVFN **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
38B (907)	**Symbol**: DORCIVNM **Meaning**: Program error. An incorrect output descriptor name was supplied to the dynamic output service. **Action**: Check the output descriptor name supplied to the dynamic output service. The first character must be an alphabetic (capitalized) character or a national character (#, @, or $). The remaining characters must be alphabetic (capitalized) characters, national characters (#, @, or $), or a number. There can be no intervening blanks. Alternatively, the name can be all (binary) zeros, indicating that the system should generate a name for this request. Correct the problem and retry the dynamic output invocation.
38C (908)	**Symbol**: DORCIVRZ **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
38D (909)	**Symbol**: DORCIVDZ **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
38E (910)	**Symbol**: DORCIVHB **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
38F (911)	**Symbol**: DORCIVTU **Meaning**: Program error. No text units were specified. Either the pointer to the text unit pointer list was zero, or all the text unit pointers were zero. **Action**: Determine why no text units were specified. Correct the problem and retry the dynamic output invocation.
390 (912)	**Symbol**: DORCP0C4 **Meaning**: Program error. An 0C4 ABEND occurred when the system referenced the parameter list. The parameter list is generated by the list form of the OUTADD macro. **Action**: Check to see if the parameter list has the same storage key as the program that uses the execute form of the macro to invoke dynamic output. Also check to see if your program passed a bad pointer or address, or if your program failed to set the leftmost bit in the last pointer of the text unit pointer list.
391 (913)	**Symbol**: DORCNZR0 **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.

Figure 27 (Page 5 of 5). Reason Codes for Return Code 0C	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
394 (916)	**Symbol**: DORCREON **Meaning**: Program error. There is an error in the use of the OUTADD macro. The execute form of the OUTADD macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTADD invocation.
409 (1033)	**Symbol**: DORCALTP **Meaning**: Program error. Dynamic output processing has determined that the length of the text unit pointer list has changed during the processing of this dynamic output request. A possible reason for this is that the application is running in a multitasking environment, where the text unit pointers may be in shared storage between multiple tasks. Another task could be altering this shared storage concurrently. **Action**: Review the design of the application and determine if additional tasks running concurrently with this dynamic output request could be altering shared storage (the text unit pointer list). If a change can be made to eliminate the storage alteration by another task, retry the dynamic output request.

Reason Codes for Return Code 10

OUTADD returns a return code of 10 for system ABENDs that occurred during dynamic output OUTADD processing.

The table below documents the reason codes that can occur when the OUTADD macro returns with a return code of 10 (in GPR15). The symbol for return code 10 is DOSYSERR in macro `IEFDORC`. The “symbol” field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro `IEFDORC`.

Figure 28. Reason Codes for Return Code 10	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
700 (1792)	**Symbol**: DORCABND **Meaning**: System error. An ABEND occurred in the dynamic output control routine. **Action**: Record the return and reason code, and optionally make a copy of your application. Give this information to your system programmer to supply to the appropriate IBM support personnel.
701 (1793)	**Symbol**: DORCSJAB **Meaning**: System error. An ABEND occurred during this dynamic output request. **Action**: Record the return and reason code, and optionally make a copy of your application. Give this information to your system programmer to supply to the appropriate IBM support personnel.
702 (1794)	**Symbol**: DORCXABD **Meaning**: System error. An ABEND occurred in the dynamic output installation exit. **Action**: Notify your system programmer. The dynamic output installation exit might have a logic error. If a change is made to the installation exit, you can retry the dynamic output invocation.

Example

Use the execute form of the OUTADD macro to modify and execute a parameter list at symbolic location PLIST. The output descriptor is at symbolic location, DESCR2. The text unit pointer list is at symbolic location, TEXTL.

```
OUTADD  NAME=DESCR2,TEXTPTR=TEXTL,MF=(E,PLIST)
```

OUTDEL — Delete an Output Descriptor

Description

Use the OUTDEL macro to delete an output descriptor for a system output (sysout) data set. For information about using the OUTDEL macro, see “Dynamic Output” in *OS/390 MVS Programming: Authorized Assembler Services Guide*.

The OUTDEL macro has no standard form. Use the list form to generate a storage declaration for the input parameter list to dynamic output. Use the execute form to modify the parameter list and invoke dynamic output.

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
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming Requirements

- Use OUTDEL only when the output descriptor being deleted was added by the OUTADD macro.
- When you code the input data to dynamic output, all character data must be left-justified in a field of blanks. Any noncharacter input data must be right-justified in a field of binary zeroes.

Restrictions

- You can use dynamic output in a JES2 environment or a JES3 4.2.1 or later environment.
- On some systems, the system does not free the output descriptor's storage when you delete the output descriptor. Whether or not this storage is freed depends on the version of JES being used on your system. For more information, see “Dynamic Output” in *OS/390 MVS Programming: Authorized Assembler Services Guide*.

Input Register Information

Before issuing the OUTDEL 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 is returned to the calling program the GPRs contain:

Register	Contents
0	Reason code
1	Zero
2-14	Unchanged
15	Return code

© Copyright IBM Corp. 1988, 2000

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.

OUTDEL—List Form

Syntax

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

name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede OUTDEL.
OUTDEL	
ƀ	One or more blanks must follow OUTDEL.
,MF=L	

Parameters

The parameters of the list form, which are both required, are explained as follows:

name
: The list form defines the storage area to be used as the input parameter list to the OUTDEL macro. *name* specifies the symbolic address of this storage.

MF=L
: Specifies the list form of the OUTDEL macro.

Example

Use the list form of the OUTDEL macro to generate the input parameter list that is to be used by the execute form of the OUTDEL macro. Locate the parameter list at symbolic location, PARML.

```
PARML    OUTDEL  MF=L
```

OUTDEL—Execute Form

The execute form of the OUTDEL macro modifies and executes the parameter list that was built with the list form of the OUTDEL macro.

Syntax

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

name	*name*: Symbol. Begin name in column 1.
␣	One or more blanks must precede OUTDEL.
OUTDEL	
␣	One or more blanks must follow OUTDEL.
NAME=*descriptor name addr*	*descriptor name addr*: RX-type address or register (2)-(12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2)-(12).

Parameters

The parameters are explained as follows:

NAME=*descriptor name addr*
Specifies the address of an 8-character field. This field contains the name of the output descriptor that is to be deleted.

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

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

ABEND Codes

OUTDEL might abnormally terminate with abend codes X'76D' or X'054'. See *OS/390 MVS System Codes* for explanations and programmer responses.

Return and Reason Codes

When control returns to the caller, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code. Note that when you invoke dynamic output to delete an output descriptor, the system has deleted the descriptor only if the return code is zero.

For programmers who want them, symbolic names for reason codes are available in the IEFDORC mapping macro. (See *OS/390 MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.) Reason codes are four bytes long.

Figure 29 (Page 1 of 2). Return and Reason Codes for the OUTDEL Macro

Return Code	Reason Code	Meaning and Action
00	00	**Meaning**: The dynamic output request is successful. **Action**: None.
04	400-407	**Meaning**: The dynamic output request failed because of an error in dynamic output or elsewhere in the system. **Action**: None.
08	380-38B 391 500-503	**Meaning**: The dynamic output request failed. There is an error in an operating system program. **Action**: Contact the system programmer.

Figure 29 (Page 2 of 2). Return and Reason Codes for the OUTDEL Macro		
Return Code	**Reason Code**	**Meaning and Action**
08	00 6000-7FFF	**Meaning**: The dynamic output request failed. The request was denied by your installation. These reason codes are defined by your installation. **Action**: None.
0C	30B-394	**Meaning**: The dynamic output request failed because the parameter list is incorrectly initialized. **Action**: None.
10	700-702	**Meaning**: An abend occurred in the system. **Action**: Consult the system programmer.

For programmers who want them, symbolic names for reason codes are available in the IEFDORC mapping macro. (See *OS/390 MVS Data Areas, Vol 2 (DCCB-ITZYRETC)*.) Reason codes are four bytes long.

Reason Codes for Return Code 04

The table below documents the reason codes that can occur when the OUTDEL macro returns with a return code of 04 (in GPR15). The symbol for return code 04 is DOENVERR in macro `IEFDORC`. The "symbol" field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro `IEFDORC`.

Figure 30. Reason Codes for Return Code 04	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
400 (1024)	**Symbol**: DORCGET1 **Meaning**: Environmental error. A GETMAIN failed during processing of the dynamic output request. **Action**: Attempt to allow a larger region for the application and retry the dynamic output request.
402 (1026)	**Symbol**: DORCNDES **Meaning**: Program error. For a delete request, the output descriptor does not exist or it has already been deleted. **Action**: This can be a logic error in the application. Check the program logic for conditions that could lead to an incorrect (nonexistent) output descriptor name being specified on the delete request.
403 (1027)	**Symbol**: DORCBTCH **Meaning**: Program error. The delete request attempted to delete a JCL-specified output descriptor. A JCL-specified output descriptor cannot be deleted with a dynamic output delete request. **Action**: This can be a logic error in the application. Check the program logic for conditions that could lead to an incorrect output descriptor name being specified on the delete request.
404 (1028)	**Symbol**: DORCESTA **Meaning**: System error. During dynamic output processing, the system was unable to establish a recovery environment. **Action**: Determine if your program is doing something that prevents a recovery environment from being established. The most likely reason for this error is that your program is causing all available storage to be used. If the problem cannot be attributed to your program, record the return and reason code, and optionally make a copy of your application. Give this information to your system programmer to supply to the appropriate IBM support personnel.
407 (1031)	**Symbol**: DORCGET2 **Meaning**: Environmental or system error. A GETMAIN request failed during processing of the dynamic output request. **Action**: Attempt to allow a larger region for the application and retry the dynamic output request.

Reason Codes for Return Code 08

The table below documents the reason codes that can occur when the OUTDEL macro returns with a return code of 08 (returned in GPR15). The symbol for return code 08 is DOREQDNY in macro IEFDORC. The "symbol" field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro IEFDORC.

Figure 31 (Page 1 of 4). Reason Codes for Return Code 08

Reason Code Hexadecimal (Decimal)	Meaning and Action
0 (0)	**Symbol**: (none defined) **Meaning**: Program error. The dynamic output request was denied by your installation. **Action**: The meaning of this reason code is defined by your installation. Your installation should be able to make recommendations for altering the dynamic output request to conform to installation standards.
380 (896)	**Symbol**: DORCLNIV **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Dynamic output processing detected an incorrect parameter list length. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNLEN field was incorrectly set and correct the problem.
381 (897)	**Symbol**: DORCNZF1 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the unused bits in the DOCNFNC1 field are not zero and correct the problem.
382 (898)	**Symbol**: DORCNZF2 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the unused bits in the DOCNFNC2 field are not zero and correct the problem.
383 (899)	**Symbol**: DORCNZR1 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNRSV1 reserved field is not zero and correct the problem.
384 (900)	**Symbol**: DORCNZR2 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNRSV2 reserved field is not zero and correct the problem.

Figure 31 (Page 2 of 4). Reason Codes for Return Code 08

Reason Code Hexadecimal (Decimal)	Meaning and Action
385 (901)	**Symbol**: DORCIVID **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A portion of the dynamic output parameter list was improperly set. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNID field was not properly initialized and correct the problem.
386 (902)	**Symbol**: DORCIVVR **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. The DOCNVERS field contains an incorrect parameter list version. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNVERS field was not properly initialized and correct the problem.
387 (903)	**Symbol**: DORCNOFN **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A portion of the dynamic output parameter list was improperly set. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Either the DOCNNEW or the DOCNDEL bit must be set in the DOCNFNC1 field. Determine why no function bit (create or delete) was set and correct the problem.
308 (904)	**Symbol**: DORCIVFN **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A portion of the dynamic output parameter list was improperly set. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Only one function bit (either DOCNNEW or DOCNDEL) should be set in the DOCNFNC1 field. Determine why more than one function bit (create or delete) was set and correct the problem.
389 (905)	**Symbol**: DORCIVTP **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A text unit pointer was specified for a delete request. The installation exit maps the dynamic output parameter list (macro IEFDOCNP); the text unit pointer was specified in field DOCNTXTP.
38A (906)	**Symbol**: DORCIVEQ **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. A conditional ENQ indicator was specified for a delete request. The installation exit maps the dynamic output parameter list (macro IEFDOCNP); the conditional ENQ bit was specified using DOCNCENQ in field DOCNFNC2. Determine why a conditional ENQ indicator was specified and correct the problem.

Figure 31 (Page 3 of 4). Reason Codes for Return Code 08

Reason Code Hexadecimal (Decimal)	Meaning and Action
38B (907)	**Symbol**: DORCIVNM **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Dynamic output processing received an incorrect output descriptor name from the installation exit. Check the output descriptor name returned to the dynamic output service. The first character must be an alphabetic (capitalized) character or a national character (#, @, or $). The remaining characters must be alphabetic (capitalized) characters, national characters (#, @, or $), or numbers. There can be no intervening blanks. The installation exit maps the dynamic output parameter list (macro IEFDOCNP); the output descriptor name is in the DOCNNAME field.
391 (913)	**Symbol**: DORCNZR0 **Meaning**: System error. **Action**: Contact your system programmer. Provide the system programmer with the return and reason code. Once the problem has been corrected, retry the dynamic output invocation. For the system programmer: The dynamic output installation exit has a logic error. Unused bits passed back from the dynamic output installation exit were not set to zero. The installation exit maps the dynamic output parameter list (macro IEFDOCNP). Determine why the DOCNRSV0 reserved field is not zero and correct the problem.
500 (1280)	**Symbol**: DORCINST **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is 8, but the reason code is not within the allowable range defined for the dynamic output installation exit. Determine why the installation exit is returning a reason code that is outside the allowable range and correct the error.
501 (1281)	**Symbol**: DORCINRC **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is zero but the reason code is not zero. Determine why the installation exit is returning a nonzero reason code but a zero return code for the dynamic output request and correct the error.
502 (1282)	**Symbol**: DORCINRT **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is not within the allowable range defined for the dynamic output installation exit. Determine why the installation exit is returning a return code that is outside the allowable range and correct the error.

Figure 31 (Page 4 of 4). Reason Codes for Return Code 08	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
503 (1283)	**Symbol**: DORCINKE **Meaning**: System error. **Action**: Contact your system programmer. Provide your system programmer with the return and reason code. Once the operating system program has been corrected, retry the dynamic output invocation. For the system programmer: There is a program error in the dynamic output installation exit. The return code from the dynamic output installation exit is zero, but the installation exit has returned a nonzero value in register 1. Although this is an OUTDEL request, when control is returned from the installation exit for an OUTADD request, register 1 contains the value of the key that is in error. Determine why the installation exit is returning a nonzero value in register 1 and a zero return code for the dynamic output request and correct the error.
6000-7FFF (24576-32767)	**Symbol**: (none defined) **Meaning**: Program error. The dynamic output request was denied by your installation. **Action**: The meaning of this reason code is defined by your installation, and indicates the reason the dynamic output request was denied. Your installation should be able to make recommendations for altering the dynamic output request to conform to installation standards.

Reason Codes for Return Code 0C

OUTDEL returns a return code of 0C for errors detected in the caller's parameters. The list form of the OUTDEL macro generates a parameter list. When the caller invokes the execute form of the OUTDEL macro, this parameter list is filled in with the parameters coded on the execute form of the OUTDEL macro. The OUTDEL service verifies this parameter list.

The table below documents the reason codes that can occur when the OUTDEL macro returns with a return code of 0C (in GPR15). The symbol for return code 0C is DOINVPRM in macro IEFDORC. The “symbol” field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro IEFDORC.

Figure 32 (Page 1 of 4). Reason Codes for Return Code 0C	
Reason Code Hexadecimal (Decimal)	**Meaning and Action**
30B (779)	**Symbol**: DORCIVNP **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
30C (780)	**Symbol**: DORCIVLN **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
311 (785)	**Symbol**: DORCMTUP **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.

Figure 32 (Page 2 of 4). Reason Codes for Return Code 0C

Reason Code Hexadecimal (Decimal)	Meaning and Action
380 (896)	**Symbol**: DORCLNIV **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
381 (897)	**Symbol**: DORCNZF1 **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
382 (898)	**Symbol**: DORCNZF2 **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
383 (899)	**Symbol**: DORCNZR1 **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
384 (900)	**Symbol**: DORCNZR2 **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
385 (901)	**Symbol**: DORCIVID **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
386 (902)	**Symbol**: DORCIVVR **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.

Figure 32 (Page 3 of 4). Reason Codes for Return Code 0C

Reason Code Hexadecimal (Decimal)	Meaning and Action
387 (903)	**Symbol**: DORCNOFN **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
388 (904)	**Symbol**: DORCIVFN **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
389 (905)	**Symbol**: DORCIVTP **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
38A (906)	**Symbol**: DORCIVEQ **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
38B (907)	**Symbol**: DORCIVNM **Meaning**: Program error. Dynamic output processing received an invalid output descriptor name. **Action**: Check the output descriptor name supplied to the dynamic output service. The first character must be an alphabetic (capitalized) character or a national character (#, @, or $). The remaining characters must be alphabetic (capitalized) characters, national characters (#, @, or $), or numbers. There can be no intervening blanks. Correct the problem and retry the dynamic output invocation.
38C (908)	**Symbol**: DORCIVRZ **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
38D (909)	**Symbol**: DORCIVDZ **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.

Figure 32 (Page 4 of 4). Reason Codes for Return Code 0C

Reason Code Hexadecimal (Decimal)	Meaning and Action
38E (910)	**Symbol**: DORCIVHB **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
390 (912)	**Symbol**: DORCP0C4 **Meaning**: Program error. An 0C4 ABEND occurred when the system referenced the parameter list The parameter list is generated by the list form of the OUTDEL macro. **Action**: Check to see if the parameter list has the same storage key as the program that uses the execute form of the macro to invoke dynamic output. Also check to see if your program passed a bad pointer or address.
391 (913)	**Symbol**: DORCNZR0 **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
392 (914)	**Symbol**: DORCONEU **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
393 (915)	**Symbol**: DORCREUS **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.
394 (916)	**Symbol**: DORCREON **Meaning**: Program error. There is an error in the use of the OUTDEL macro. The execute form of the OUTDEL macro creates assembler instructions in your program. The list form of the macro reserves storage for the input parameter list that is used by the execute form of the macro. An error has been detected in the assembler instructions or in the parameter list storage. Another task might have altered the assembler instructions' storage or the parameter list's storage. **Action**: Correct the problem and retry the OUTDEL invocation.

Reason Codes for Return Code 10

OUTDEL returns a return code of 10 for system ABENDS that occurred during dynamic output OUTDEL processing.

The table below documents the reason codes that can occur when the OUTDEL macro returns with a return code of 10 (in GPR15). The symbol for return code 10 is DOSYSERR in macro IEFDORC. The “symbol” field in the following table gives the symbolic name for the different reason code values. These symbols can be found in macro IEFDORC.

Figure 33. Reason Codes for Return Code 10

Reason Code Hexadecimal (Decimal)	Meaning and Actions
700 (1792)	**Symbol**: DORCABND **Meaning**: System error. An ABEND occurred in the dynamic output control routine. **Action**: Record the return and reason code, and optionally make a copy of your application. Give this information to your system programmer to supply to the appropriate IBM support personnel.
701 (1793)	**Symbol**: DORCSJAB **Meaning**: System error. An ABEND occurred during this dynamic output request. **Action**: Record the return and reason code, and optionally make a copy of your application. Give this information to your system programmer to supply to the appropriate IBM support personnel.
702 (1794)	**Symbol**: DORCXABD **Meaning**: System error. An ABEND occurred in the dynamic output installation exit. **Action**: Notify your system programmer. The dynamic output installation exit might have a logic error. If a change is made to the installation exit, it might be possible to retry the dynamic output invocation.

Example

Use the execute form of the OUTDEL macro to modify and execute a parameter list at symbolic location PLIST. The output descriptor is at symbolic location, DESCR2.

```
OUTDEL  NAME=DESCR2,MF=(E,PLIST)
```

PCLINK — Stack, Unstack, or Extract Program Call Linkage Information

Description

Note: IBM recommends the use of stacking PC routines instead of basic PC routines; stacking PC routines use system-provided linkage rather than issuing PCLINK to save and restore the caller's environment.

Routines that receive control as a result of a basic PC instruction use the PCLINK macro to provide a standardized method of maintaining basic PC linkage information. PCLINK has three forms:

- PCLINK STACK saves some of the environment when a routine gets control as a result of a basic PC instruction.
- PCLINK UNSTACK restores that environment before the routine issues a PT instruction to return control to the calling routine.
- PCLINK EXTRACT retrieves information from the environment that PCLINK STACK saved.

See *OS/390 MVS Programming: Extended Addressability Guide* for information about basic and stacking PC routines, the instructions they can use, and the environmental information that PCLINK saves and restores.

STACK Option of PCLINK

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
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

Your program must not change registers 13-4 between receiving control and the time of issuing PCLINK.

Input Register Information

Before issuing the STACK option of the PCLINK 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, 2000

Output Register Information

On completion of PCLINK STACK, the registers are as follows:

Register	Contents
0-1	Unchanged
2	Bits 0-23 contain bits 8-31 from register 2 at the time the PCLINK macro was issued. Bits 24-31 contain the PCLINK caller's PSW key.
3-4	Unchanged
5	Used as a work register by the system
6-7	Unchanged
8-12	Unchanged if SAVE=YES, used as work registers by the system if SAVE=NO
13	Zero, to ensure that the first save area created after the basic PC does not point to a previous save area.
14	Stack token to uniquely identify the stack entry created. This token is required for the UNSTACK and EXTRACT forms of PCLINK.
15	Unchanged

Performance Implications

Processing is more efficient if SAVE=NO is specified.

Syntax

The STACK option of the PCLINK macro is written as follows:

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

b                          One or more blanks must precede PCLINK.

PCLINK

b                          One or more blanks must follow PCLINK.
```

```
STACK

   ,INKEY=ZERO

   ,OUTKEY=CALLER          Default: OUTKEY=CALLER
   ,OUTKEY=ZERO
   ,OUTKEY=KEYn            n: Any valid PSW key value from 0-F.

   ,SAVE=YES               Default: SAVE=YES
   ,SAVE=NO

   ,RELATED=value          value: Any valid macro keyword specification.
```

Parameters

The parameters are explained as follows:

STACK
Saves some of the environment when a routine gets control as a result of a basic PC instruction.

,INKEY=ZERO
Specifies that the PSW key is zero upon entry to PCLINK. If this parameter is not specified, the PSW key is temporarily changed to zero.

,OUTKEY=CALLER
,OUTKEY=ZERO
,OUTKEY=KEY*n*
Specifies the setting of the PSW key after the PCLINK macro has completed. Specifying CALLER causes the PSW key to be restored to the value it had on entry. Specifying ZERO sets the PSW key to zero. Specifying a key value indicates a specific value for the key. You may specify any key value from zero to F.

,SAVE=YES
,SAVE=NO
Specifies whether (YES) or not (NO) to preserve registers 8 - 12. The save area used is different from the area addressed by register 13. SAVE=YES is the default.

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

ABEND Codes

052
053

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

UNSTACK Option of PCLINK

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state, any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
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

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the UNSTACK option of the PCLINK 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.

Performance Implications

Processing is more efficient if UNSTACK,SAVE=NO,THRU is specified separately for each stack element to be dequeued rather than one request for several elements.

Syntax

The UNSTACK option of the PCLINK macro is written as follows:

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

b                          One or more blanks must precede PCLINK.

PCLINK

b                          One or more blanks must follow PCLINK.

UNSTACK

,THRU=(reg)                reg: Register (0) - (15).
,TO=(reg)
,PURGE=YES

   ,INKEY=ZERO

   ,OUTKEY=STACK           Default: OUTKEY=STACK
   ,OUTKEY=ZERO

   ,SAVE=YES               Default: SAVE=YES
   ,SAVE=NO

   ,ERRET=addr             addr: RX-type address or register (0) - (13) or (15).

   ,RELATED=value          value: Any valid macro keyword specification.
```

Parameters

The parameters are explained as follows:

UNSTACK
Restores the environment before the routine issues a PT instruction to return control to the calling routine.

,THRU=*(reg)*
Specifies that the stack element identified by the token contained in the specified register, as well as all more recently stacked elements, are to be removed from the requestor's stack. The stack element specified by the token is used to restore registers. If the system cannot process the request, the routine specified by the ERRET parameter gets control; if the ERRET parameter is not specified, the requestor is abnormally terminated.

When a PCLINK UNSTACK,THRU is completed, the PSW program mask is restored from the stack element identified by the token and the registers are as follows:

Register	Contents
0-1	Unchanged
2	Bits 24-27 contain the PSW key from the stack element identified by the token
3	As saved by PCLINK STACK
4-7	Unchanged
8-12	Unchanged if SAVE=YES is specified, used as work registers by the system if SAVE=NO is specified

13-14	As saved by PCLINK STACK
15	Unchanged

,TO=*(reg)*

Specifies that all stack elements stacked more recently than the element identified by the token contained in the specified register are to be removed from the stack. The element identified by the token remains on the stack. If the system cannot process the request, the routine specified by the ERRET parameter gets control; if the ERRET parameter is not specified, the requestor is abnormally terminated.

Use the TO parameter for stack cleanup in an FRR or ESTAE retry routine or in an FRR that is going to retry.

When a PCLINK UNSTACK,TO is completed, the registers are as follows:

Register	Contents
0-1	Used as work registers by the system
2	Unchanged if INKEY=ZERO is specified and ERRET is not specified; otherwise, PSW key of PCLINK caller
3-7	Unchanged
8-12	Unchanged if SAVE=YES is specified, used as work registers by the system if SAVE=NO is specified
13	Unchanged
14-15	Used as work registers by the system

,PURGE=YES

Specifies that each stack element is to be freed until no more exist on the requestor's stack. Any element that resides in a terminated address space as well as elements stacked prior to it are not freed, but the stack pointer indicates an empty stack and the PCLINK request returns normally to the caller.

The ERRET parameter cannot be used with PURGE.

When the PCLINK UNSTACK,PURGE is completed, the registers are as follows:

Register	Contents
0-1	Used as work registers by the system
2	Unchanged if INKEY=ZERO is specified; otherwise, PSW key of PCLINK caller
3-7	Unchanged
8-12	Unchanged if SAVE=YES is specified, used as work registers by the system if SAVE=NO is specified
13	Unchanged
14-15	Used as work registers by the system

,INKEY=ZERO

Specifies that the PSW key is zero on entry to PCLINK. If this parameter is not specified, the key is temporarily changed to zero.

,OUTKEY=STACK
,OUTKEY=ZERO

Specifies the setting of the PSW key after the PCLINK request is completed. Specifying OUTKEY=ZERO returns to the caller in key zero. Specifying OUTKEY=STACK restores the key to the value contained in the stack element identified by token. OUTKEY=STACK is the default.

This parameter is valid only with PCLINK UNSTACK,THRU.

,SAVE=YES
,SAVE=NO

Specifies whether (YES) or not (NO) registers 8 - 12 are to be preserved. The save area used for these registers is not the area pointed to by register 13.

,ERRET=*addr*

Specifies the address of an exit routine to be given control if PCLINK UNSTACK encounters an error. ERRET is valid only with the TO and THRU parameters.

The ERRET exit routine receives control in the addressing mode of the caller of PCLINK. When an ERRET exit routine gets control, the cross memory state is the same as when the PCLINK macro was issued. The registers are as follows:

Register	Contents
0-1	Used as work registers by the system
2	PSW key of PCLINK caller
3	Used as a work register by the system
4-7	Unchanged
8-12	Unchanged if SAVE=YES is specified, used as work registers by the system if SAVE=NO is specified
13	Used as a work register by the system
14	The token passed as input
15	**4** - stack was empty **8** - input token is invalid **12** - an address on the queue is invalid **16** - An ASID on the queue is invalid **20** - Unknown error

,RELATED=*value*

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

ABEND Codes

052
053

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

EXTRACT Option of PCLINK

PCLINK EXTRACT modifies registers 0, 1, 14, and 15. If ALL=YES is specified, registers 13-4 are also modified.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state or PSW key 0 or PKM key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in primary address space

Programming Requirements

Your program must have addressability to the address space from which PCLINK STACK was issued for the current stack element.

Restrictions

None.

Input Register Information

Before issuing the EXTRACT option of the PCLINK 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.

Performance Implications

None.

Syntax

The EXTRACT option of the PCLINK macro is written as follows:

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

EXTRACT	
,TOKEN=*(reg)*	*reg*: Register (0) - (15).
,ALL=YES	
,SVAREA=*(reg)*	
,RETADR=*(reg)*	
,PARM15=*(reg)* ,PARM0=*(reg)* ,PARM1=*(reg)*	
,KEY=*(reg)*	
,ASID=*(reg)*	
,LP=*(reg)*	
,ENTRY=*(reg)*	
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

EXTRACT
Retrieves information from the saved environment.

,TOKEN=*(reg)*
Specifies a register that contains a 32-bit stack token identifying the most recently stacked element.

,ALL=YES
Specifies that all information stored in the stack element identified by the token is to be extracted. The stored information is placed into the same registers (registers 13, 15, and 0-4) it was in when PCLINK STACK was issued. Registers 5 and 14 are not restored.

,SVAREA=*(reg)*
Specifies a register into which the address of the program call issuer's save area is to be placed.

,RETADR=*(reg)*
Specifies a register into which the AMODE (in which control is to be returned), the return address, and PSW problem state bit are to be placed. These occupy bits 0,1-30, and 31, respectively.

,PARM15=*(reg)*
,PARM1=*(reg)*
,PARM0=*(reg)*
Specifies a register into which the contents of register 15 (PARM15), register 1 (PARM1), or register 0 (PARM0) at the time PCLINK STACK was issued are to be placed.

,KEY=*(reg)*
Specifies a register into which the basic PC issuer's PSW key is to be placed. The key occupies bit positions 24-27.

,ASID=*(reg)*
Specifies a register into which the basic PC issuer's PSW key mask (bits 0-15) and ASID (bits 16-32) are to be placed.

,LP=*(reg)*
Specifies a register into which the latent parameter list address is to be placed.

,ENTRY=*(reg)*
Specifies a register into which the contents of register 5 as established by the PCLINK STACK macro are to be placed. Bit 0 of the register used by the ENTRY parameter specifies the addressing mode of the program call routine that issued the PCLINK macro.

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

ABEND Codes

052
053

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

PGANY — Page Anywhere

Description

Note: IBM recommends that you use the PGSER macro rather than PGANY.

Some fixed pages are assigned within the first 16 megabytes of storage. The system assumes that once a page has been fixed, it is likely to be fixed again. The next time that page is loaded, the system tries to put it in the first 16 megabytes in anticipation of a fix. Use the PGANY macro to indicate to the system that no further page fixes are planned for a particular page and that the next time the page is loaded, the system can put it anywhere.

Input Register Information

Entry is by means of an SVC. The caller can be in either problem or supervisor state and must not hold any locks.

Output Register Information

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

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

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

Syntax

The PGANY macro is written as follows:

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

L,LA=*list addr*	*list addr*: RX-type address or register (1) or (2) - (12).
R,A=*start addr*	*start addr*: RX-type address or register (1), (2) - (12).
,EA=*end addr*	*end addr*: RX-type address or register (15), (2) - (12). **Note**: Cannot be specified unless R is specified. **Default**: EA=start addr + 1.

© Copyright IBM Corp. 1988, 2000

Parameters

The parameters are explained as follows:

L Specifies that the virtual subarea list (VSL) is being supplied with this request. (See "Input to Page Services" in *OS/390 MVS Programming: Authorized Assembler Services Guide* for a description of the virtual subarea list.)

,LA=*list addr*
Specifies the address of the virtual subarea list.

R Specifies that the necessary parameters will be passed in registers. A virtual subarea list is not being supplied.

,A=*start addr*
Specifies the address of the start of the virtual area.

,EA=*end addr*
Specifies the end + 1 byte of the virtual area. If this parameter is not coded, the default is the start address + 1.

Note: *start addr* and *end addr* must be located in 24-bit addressable storage.

Return and Reason Codes

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

Figure 34. Return Codes for the PGANY Macro

Return Code	Meaning
00	**Meaning**: Operation completed normally.
04	**Meaning**: Parameter error, X'171' abend, operation terminated because of invalid address in VSL entry.
10	**Meaning**: Parameter error, X'171' abend, operation terminated abnormally because the VSL list was invalid.
14	**Meaning**: Environmental error, X'028' abend.

For return codes 04 and 10, registers are loaded before the abend as follows:

R0	Used as a work register by the macro
R1	Abend code
R2-R10	Used as a work register by the macro
R11	Address of input VSL list or 0 for R-form
R12	0 (ECB address =0)
R13-R14	Current VSL entry being processed
R15	Return code

PGFIX — Fix Virtual Storage Contents

Description

Note: IBM recommends that you use the PGSER macro rather than PGFIX.

The PGFIX macro makes virtual storage areas, below 16 megabytes, resident in central (also called real) storage and ineligible for page-out while the requesting task's address space is swapped into central storage. PGFIX ignores requests to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by central storage below 16 megabytes. A subsequent PGFREE is effective only if issued by the same task. The PGFIX function is available only to authorized users.

PGFIX does not prevent pages from being paged out when an entire address space is swapped out of central storage. Consequently, when using the PGFIX macro, you cannot assume a constant real address mapping for fixed pages that are susceptible to swapping.

Syntax

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

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

R L	
,LA=*list addr*	*list addr*: A-type address, or register (1) or (2) - (12).
,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
,LONG=Y ,LONG=N	**Default**: LONG=Y
,RELEASE=N ,RELEASE=Y	**Default**: RELEASE=N **Note**: RELEASE=Y may only be specified with EA above.
,RELATED=*value*	*value*: Any valid macro keyword specification.

© Copyright IBM Corp. 1988, 2000

Parameters

The parameters are explained as follows:

R Specifies that no parameter list is being supplied with this request.

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 "Input to Page Services" in *OS/390 MVS Programming: Authorized Assembler Services Guide* for a description of the VSL.

,A=*start addr*
Specifies the start address of the virtual area to be fixed.

Note: *start addr* must be located in 24-bit addressable storage.

,ECB=*ecb addr*
Specifies the address of the ECB that is used to signal event completion. If the ECB address specified is zero, (ECB=0 or ECB=(register) where the contents of the register specified is 0), the fix request is completely satisfied before control is returned.

Note: If the user intends to wait on the ECB as part of an ECB list, he must ensure that the list and associated ECBs are fixed in central storage before issuing the WAIT. The PGFIX service routine ensures that the specified ECB is fixed.

,EA=*end addr*
Specifies the end address + 1 of the virtual area to be fixed.

Note: *end addr* must be located in 24-bit addressable storage.

,LONG=Y
,LONG=N
Specifies that the relative real time duration anticipated for the fix is long (Y) or short (N).

,RELEASE=N
,RELEASE=Y
Specifies that the contents of the virtual area is to remain intact (N) or be released (Y) before the fix is done.

,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.

Return and Reason Codes

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

Figure 35. Return Codes for the PGFIX Macro

Return Code	Meaning
00	**Meaning**: Operation completed normally; ECB posted complete.
04	**Meaning**: Operation abnormally terminated with a X'171' abend. Operation incomplete because of invalid address virtual subarea list entry; ECB posted complete. See *OS/390 MVS System Codes* for a complete description of the register contents after a X'171' abend.
08	**Meaning**: Operation proceeding; ECB will be posted when all requested pages are fixed in central storage.
10	**Meaning**: Operation abnormally terminated with a X'171' abend. Virtual subarea list entry or ECB address invalid; no ECB is posted. See *OS/390 MVS System Codes* for a complete description of the register contents after a X'171' abend.

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.
4 - operation incomplete because of invalid address in VSL entry.

If the return code issued is 8, the ECB is posted asynchronously when paging I/O has completed, with code:

0 - operation completed successfully.
4 - operation incomplete because of paging error; requesting TCB will be abnormally terminated.

The ECB code is posted in the low-order 3 bytes of the ECB, and is right-justified.

Example 1

Fix a single byte of virtual storage addressed by register 3. Note that the full 4096-byte page containing the specified byte is actually fixed. The storage is long fixed.

```
PGFIX R,A=(R3),ECB=(R5)
```

Example 2

Fix virtual storage without using a virtual subarea list. Storage is long fixed.

```
PGFIX R,A=(R3),EA=(R4),ECB=ECB1
```

Example 3

Fix, but not long-fix, virtual storage, and ensure that the pages fully included in the address range are forfeited before fixing the area specified by registers 3 and 4.

```
PGFIX R,A=(R3),EA=(R4),ECB=(R5),LONG=N,RELEASE=Y
```

PGFIXA — Fix Virtual Storage Contents

Description

Note: IBM recommends that you use the PGSER macro rather than PGFIXA.

The PGFIXA macro makes virtual storage areas, below 16 megabytes, resident in central (also called real) storage and ineligible for page-out while the requesting task's address space is swapped into central storage. The PGFIXA function is available only to key zero and supervisor state users. The PGFIXA macro executes short-term, synchronous page fixes. The preferred area(s) of storage are intended for long term page fixes. A long term page fix in the V=R or nonpreferred areas may delay V=R functions or CONFIG STORAGE commands. All fix processing is assumed to be short-term and is complete when control is returned to the issuer of the macro.

PGFIXA does not prevent pages from being paged out when an entire address space is swapped out of central storage. Consequently, when using the PGFIXA macro, you cannot assume a constant real address mapping for fixed pages that are susceptible to swapping.

Output

If the PGFIXA is successful, control is returned enabled to the user, all pages are fixed, and register 15 contains a return code of zero.

If the PGFIXA is unsuccessful, the user will be abended with a system completion code of X‘171’ or a system complete code of X‘028’. For X‘171’ abends, all pages processed up to, but not including the page causing the error, will be fixed. Register 10 will contain the address of the pages in error when the abend is issued. No pages will be fixed in the event of a X‘028’ abend.

Restrictions

Use of the PGFIXA macro is subject to the following restrictions:

- Can be used only for short term synchronous fixes.
- The user must be in supervisor state with a protection key of zero.
- The user must not hold any spin locks.
- The program mask byte in the PSW is zero and interrupts are enabled upon return from the PGFIXA.
- The user is responsible for freeing any pages fixed via the PGFIXA. A corresponding PGFREEA macro should be issued. In addition, an FRR should be established during the period where fixes are outstanding. The FRR should free the frames in case there is an unexpected error.
- DSECTs for the IHAPSA, CVT, and IHAPVT must be provided.
- The user must ensure that the end address is greater than or equal to the start address.
- The SAVE keyword can only be used with TYPE=R.

Syntax

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

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

© Copyright IBM Corp. 1988, 2000

PGFIXA

ƀ	One or more blanks must follow PGFIXA.
,TYPE=L ,TYPE=R	**Default**: TYPE=R
,SAVE=YES ,SAVE=NO	**Default**: SAVE=YES

Parameters

The parameters are explained as follows:

TYPE=L
TYPE=R
Specifies the type of input. When L is specified, register 1 is to contain the address of a virtual subarea list (VSL) fixed in storage. (See the topic "Input to Page Services" in *OS/390 MVS Programming: Authorized Assembler Services Guide* for a description of the VSL.) By specifying TYPE=L, registers 1 through 13 are saved. If TYPE=R is specified, then register 1 contains the address of the first byte to be fixed in a contiguous range and register 2 contains the address of the last byte to be fixed (actual end address). When TYPE=R is specified, the registers saved depend upon what is specified on the SAVE parameter.

Note: All other users of the PGFIX, PGFIXA (TYPE=L), and PGFREEA macros must specify the actual end address plus one.

,SAVE=YES
,SAVE=NO
Specifies the registers to be saved for TYPE=R. Registers 1 through 13 are saved if SAVE=YES is specified or if the default is taken. Registers 2 through 10 are saved if SAVE=NO is specified.

Example 1

Use PGFIXA to fix virtual storage without using a virtual subarea list. Registers 2 through 10 will be saved.

```
FIX1 PGFIXA TYPE=R,SAVE=NO
```

Example 2

Use PGFIXA to fix virtual storage using a virtual subarea list. Registers 1 through 13 will be saved.

```
FIX2 PGFIXA TYPE=L
```

PGFREE — Free Virtual Storage Contents

Description

Note: IBM recommends that you use the PGSER macro rather than PGFREE.

The PGFREE macro makes virtual storage pages, below 16 megabytes, that were fixed via the PGFIX macro eligible for page-out. The PGFREE function is available only to authorized users. PGFREE must be issued by the same task that issued the PGFIX, otherwise PGFREE has no effect.

Note: A fixed page is not considered pageable until the number of PGFREEs issued for the page is equal to the number of PGFIXes previously issued for that page. That is, a page is not automatically made pageable as the result of issuing a PGFREE macro.

Syntax

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

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

L	
,LA=*list addr*	*list addr*: A-type address, or register (1) or (2) - (12).
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
,ANYWHER=N ,ANYWHER=Y	**Default**: ANYWHER=N
,RELEASE=N ,RELEASE=Y	**Default**: RELEASE=N **Note**: RELEASE=Y may only be specified with EA above.
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

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 “Input to Page Services” in *OS/390 MVS Programming: Authorized Assembler Services Guide* for a description of the VSL.

© Copyright IBM Corp. 1988, 2000

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 freed.

Note: *start addr* must be located in 24-bit addressable storage.

,ECB=*ecb addr*
Specifies the address of the ECB that was used in a prior PGFIX request. This parameter is used if there is any possibility that the ECB for the previously issued PGFIX was not posted complete.

,EA=*end addr*
Specifies the end address + 1 of the virtual area to be freed.

Note: *end addr* must be located in 24-bit addressable storage.

,ANYWHER=N
,ANYWHER=Y
On subsequent page-ins, assign real frames below 16 megabytes in anticipation of a page fix (N) or on subsequent page-ins, assign real frames anywhere (Y). The ANYWHER option takes effect only when the page fix count goes to zero. The default is ANYWHER=N.

,RELEASE=N
,RELEASE=Y
Specifies that the contents of the virtual area is to remain intact (N) or be released (Y).

,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.

Return and Reason Codes

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

Figure 36. Return Codes for the PGFREE Macro

Return Code	Meaning
00	**Meaning**: Operation completed normally.
04	**Meaning**: Operation abnormally terminated. Operation incomplete because of invalid address in virtual subarea list entry.
10	**Meaning**: Operation abnormally terminated. Virtual subarea list entry or ECB address invalid.

Example 1

Free the storage in Example 1 of standard-form PGFIX.

```
PGFREE R,A=(R3)
```

Example 2

Free the storage in Example 2 of standard-form PGFIX.

```
PGFREE R,A=(R3),EA=(R4)
```

Example 3

Free the storage in Example 3 of standard-form PGFIX, and forfeit the pages fully included in the address range.

```
PGFREE R,A=(R3),EA=(R4),ECB=(R5),RELEASE=Y
```

PGFREEA — Free Virtual Storage Contents

Description

Note: IBM recommends that you use the PGSER macro rather than PGFREEA.

The PGFREEA macro makes virtual storage areas, below 16 megabytes, that were fixed by the PGFIXA macro eligible for page-out.

Syntax

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

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

No additional parameters are specified.

Restrictions

Use of the PGFREEA macro is subject to the following restrictions:

- The issuer of the PGFREEA must provide a fixed virtual subarea list (VSL) or chain of them, pointed to by register 1. For a description of the VSL, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.
- The user must be in supervisor state, protection key 0.
- The user must provide DSECTs for IHAPSA, CVT, and IHAPVT.

Output

If the PGFREEA is successful, all pages will be freed and register 15 will contain a return code of zero. If unsuccessful, all pages up to, but not including the one that caused the abend will be freed. The user will be abended with a system completion code of X'171'.

© Copyright IBM Corp. 1988, 2000

PGSER — Page Services

Description

The PGSER macro and its fast path version (see “PGSER — Fast Path Page Services” on page 179) perform the same paging services that the PGANY, PGFIX, PGFIXA, PGFREE, PGFREEA, PGLOAD, PGOUT, and PGRLSE macros perform for addresses below 16 megabytes. The PGSER macro performs these services for addresses either above or below 16 megabytes.

The syntax of the fast path version of PGSER is presented separately following the standard description.

Note: IBM recommends the use of PGSER for paging services.

The services are:

- Page fix equivalent to the PGFIX macro
- Fast path to fix virtual storage
- Page free equivalent to the PGFREE macro
- Fast path to free virtual storage
- Page load equivalent to the PGLOAD macro
- Page out equivalent to the PGOUT macro
- Page release equivalent to the PGRLSE macro
- Page anywhere equivalent to the PGANY 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 invoking PGSER with BRANCH=N are:

Minimum authorization:
- Problem state, and any key except as noted under “Restrictions” on page 172.
- To use the PROTECT and UNPROTECT options, the caller must have either PSW key 0 or a PSW key that matches the key of the storage.
- The parameters FIX and FREE are restricted to APF-authorized, key 0, or supervisor state callers. (See “Branch Entry to the PGSER Routine” in *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information about branch entry.)
- The RELEASE option of the macro is restricted to supervisor state with key 0 callers if pages in the common area are being released.

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

The requirements for the caller invoking PGSER with BRANCH=Y are:

© Copyright IBM Corp. 1988, 2000

Minimum authorization:	Supervisor state and key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or AR
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No spin 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 caller must include the IHAPVT mapping macro.
- Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the caller is executing in 31-bit addressing mode.
- 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

IBM recommends that page fixes of more than 100 pages be divided into several smaller fix requests. Large page fix requests can cause an excessive spin loop to occur.

Input Register Information

Before issuing PGSER with BRANCH=Y, the caller must ensure that GPR 13 points to a standard 18-word save area. Before issuing PGSER with BRANCH=N, 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
0-4	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) contain:

Register	Contents
0-4	Used as work registers by the system
5-13	Unchanged
14	Used as work registers by the system
15	Return Code

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

Performance Implications

None.

Syntax

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

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

R
L

,FIX
,FREE
,LOAD
,OUT
,PROTECT
,UNPROTECT
,RELEASE
,ANYWHER

,LA=*list addr*	*list addr*: RX-type address or register (1), (5) - (12) for branch entry; or register (1), (2) - (12) for SVC entry. **Note**: This parameter is valid only with L.
,A=*start addr*	*start addr*: RX-type address or register (1), (5) - (12) for branch entry; or register (1), (2) - (12) for SVC entry. **Note**: This parameter is valid only with R.
,EA=*end addr*	**Default**: EA=*start addr* *end addr*: RX-type address or register (2), (5) - (12) for branch entry; or register (15), (2) - (12) for SVC entry. **Note**: This parameter is valid only with R.
,TCB=*tcb addr*	**Default**: TCB=0 *tcb addr*: RX-type address or register (4), (5) - (12). **Note**: This parameter can be specified only if FIX, FREE, LOAD, or OUT and BRANCH=Y are specified.
,ECB=*ecb addr*	**Default**: If FREE or LOAD is specified, ECB=0. *ecb addr*: RX-type address or register (0), (5) - (12) for branch entry; or register (0), (2) - (12) for SVC entry. **Note**: This parameter is required if FIX is specified; is optional if FREE or LOAD is specified; and is not valid for OUT, RELEASE, or ANYWHER. For synchronous page fix the ECB address must be 0.
,RELEASE=Y ,RELEASE=N	**Default**: RELEASE=N **Note**: This parameter may be specified only if FIX, FREE, or LOAD is specified.
,LONG=Y ,LONG=N	**Default**: LONG=Y **Note**: This parameter may be specified only if FIX is specified.
,BACKOUT=Y ,BACKOUT=N	**Default**: BACKOUT=Y **Note**: This parameter may be specified only if FIX is specified.

,KEEPREL=Y ,KEEPREL=N	**Default**: KEEPREL=N **Note**: This parameter may be specified only if OUT is specified.
,ANYWHER=Y ,ANYWHER=N	**Default**: ANYWHER=N **Note**: This parameter may be specified only if FREE is specified.
,BRANCH=Y ,BRANCH=N	**Default**: BRANCH=N
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

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, which specifies the virtual area for which the service is to be performed. See "Input to Page Services" in *OS/390 MVS Programming: Authorized Assembler Services Guide* for a description of the PSL.

,FIX
,FREE
,LOAD
,OUT
,PROTECT
,UNPROTECT
,RELEASE
,ANYWHER
Indicates the function to be performed.

FIX specifies that the virtual storage areas are to reside in central (also called real) storage and are ineligible for page-out while the address space is swapped in. This parameter does not prevent pages from being paged out when the entire address space is swapped out of central storage. FIX will ignore a request to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by central storage below 16 megabytes. Requests for disabled reference (DREF) storage are not valid for the FIX parameter.

FREE specifies that the virtual storage areas that were previously fixed via the FIX option are eligible for page-out. A fixed page is not considered pageable until the number of FREE and FIX requests for the page are equal. Requests for disabled reference (DREF) storage are not valid for the FREE parameter.

LOAD specifies that a page-in operation is to be initiated for the virtual storage area specified, in anticipation of future needs. Requests for disabled reference (DREF) storage are not valid for the LOAD parameter.

OUT specifies that a page-out operation is to be initiated for the virtual storage area specified. Requests for disabled reference (DREF) storage are not valid for the OUT parameter.

PROTECT specifies that a range of virtual storage be made read-only. R, L, LA, A, BRANCH, 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, BRANCH, EA, and RELATED are valid keywords with the UNPROTECT option. The caller must have either key 0 or a PSW key that matches the key of the storage.

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. RELEASE is the only PGSER function that is valid for disabled reference (DREF) storage.

Use PGSER RELEASE instead of the MVCL instruction for these reasons:

- PGSER RELEASE is faster than MVCL for very large areas.
- Pages that are released through PGSER RELEASE do not occupy space in central, expanded, or auxiliary storage.

ANYWHER applies to virtual storage areas that did not specify LOC=(BELOW,ANY) or LOC=(ANY,ANY) or LOC=ANY on a GETMAIN request, that have been previously fixed, and probably will not need to be fixed again. ANYWHER specifies that the virtual storage area specified can be placed either above or below 16 megabytes central on future page-ins.

,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 last byte of the virtual area to be fixed for R requests.

,TCB=*tcb addr*
Specifies either zero or the address of the TCB to be assigned ownership of fixes for a FIX request or fixes for a FREE request. If zero is specified, no TCB is assigned ownership of the request. Cross memory callers must specify zero.

For OUT and LOAD requests, the PGSER routine associates the request with a particular TCB so that the request can be purged if the task terminates before the request is complete. For SVC entry (BRANCH=N), the PGSER routine uses the current TCB.

Note: The TCB resides in storage below 16 megabytes.

,ECB=*ecb addr*
Specifies the address of the ECB that is used to signal event completion for an asynchronous FIX or LOAD request. If the caller is in cross memory mode or if the caller requests a synchronous page fix (a FIX for which the caller is suspended until the entire FIX request is complete), the ECB must be zero (ECB=0 or ECB=(r), where (r) represents a register that contains zero).

For a FREE request, ECB specifies the address of the ECB that was used in a previous FIX request. If this parameter is specified, any pages in the previous FIX request that are not yet fixed, will not be fixed. If L is specified, the PSL chain must contain the addresses of the virtual pages in the same order in both the FREE and the previous FIX request. Also, the ECB for the FIX request will not be posted if it was not yet posted at the time of the FREE request.

If the ECB parameter is not specified on a FREE request, only the fix counts for the valid pages in storage at the time of the FREE request are decreased. This will not affect the paging activity and the posting of the ECB associated with the original FIX request.

If an ECB is supplied on a FIX or LOAD request, 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.

For all callers that supply an ECB, page services verifies that the ECB address is in an area allocated through the GETMAIN macro and if the caller is not in key 0, page services also verifies that the ECB is in the caller's protect key. You must ensure that the page containing the ECB is not freed and that the key is not altered; otherwise, page services does not post the ECB.

,RELEASE=Y
,RELEASE=N
Specifies that all the central 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 not valid and the real frames freed for reuse (N).

,LONG=Y
,LONG=N
Specifies that the relative real time anticipated for the FIX is long (Y); or that the relative real time anticipated for the FIX is short (N). (In general, the duration of a fix is long if it can be measured in seconds.)

,BACKOUT=Y
,BACKOUT=N
Specifies the procedure to follow when a nonallocated page is encountered during the processing of a FIX request. If BACKOUT=Y, all pages fixed as part of the request are freed before returning to the caller. If BACKOUT=N, the pages previously fixed as part of the request are not freed and no further processing is done before returning to the caller.

,ANYWHER=N
,ANYWHER=Y
Specifies that on subsequent page-ins, page services is to assign real frames below 16 megabytes in anticipation of a page-fix (N); or on subsequent page-ins, page services is to assign real frames anywhere (Y). The ANYWHER option takes effect only when the page-fix count goes to zero.

,BRANCH=Y
,BRANCH=N
Specifies whether this is a branch entry.

If BRANCH=Y is specified, it is a branch entry and users of this option must provide the address of an 18-word save area in GPR 13. Cross memory callers and callers in AR mode must use BRANCH=Y.

If BRANCH=N is specified, it is an SVC entry.

,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 *OS/390 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
FIX	0	**Meaning**: The operation completed normally and the ECB will not be posted. **Action**: None. If the ECB parameter was specified, do not wait on the ECB after receiving this return code because it will not be posted.
FIX	8	**Meaning**: The operation is proceeding. The ECB (if available) will be posted with X'00' when the requested pages are fixed. **Action**: None. However, if the ECB parameter was specified, issuing the WAIT macro for this ECB will allow your program to synchronize with the completion of the page fix operation.
FREE	0	**Meaning**: The operation completed normally. **Action**: None.
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 the 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 will be posted with X'00' when all page-ins are complete. **Action**: None. However, if the ECB parameter was specified, issuing the 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.
OUT	C	**Meaning**: The operation completed normally. At least one page in the requested range was not paged out. **Action**: None.
RELEASE	0	**Meaning**: The operation completed normally. **Action**: None.
ANYWHER	0	**Meaning**: The operation completed normally. **Action**: None.

Example 1

Synchronously fix the page that starts at the address given in register 1 and ends at the address given in LOADWORD. Use branch entry. No particular TCB is associated with this request. Include the IHAPVT mapping macro.

```
PGSER R,FIX,A=(1),ECB=0,EA=LOADWORD,TCB=0,BRANCH=Y
IHAPVT
```

Example 2

Free the page specified in the PSL pointed to by register 2. The ECB address is given in register 8. Use branch entry. Release all central and auxiliary storage associated with this virtual area. Do not attempt to back the area below 16 megabytes on future page-ins. Include the IHAPVT mapping macro.

```
PGSER L,FREE,LA=(2),ECB=(8),RELEASE=Y,ANYWHER=Y,BRANCH=Y
IHAPVT
```

Example 3

Load the page specified in the PSL pointed to by register 1. Supply an ECB of zero. Include the IHAPVT mapping macro.

```
PGSER L,LOAD,LA=(1),ECB=0
IHAPVT
```

Example 4

Perform a page-out for the virtual area starting at the address given in register 1 and ending at the address given in register 5. The address of the TCB is given in register 8. Use branch entry. Include the IHAPVT mapping macro.

```
PGSER R,OUT,A=(1),EA=(5),TCB=(8),BRANCH=Y
IHAPVT
```

Example 5

Perform a page-out for the virtual area specified in the PSL located at LOADWORD. Use branch entry. Include the IHAPVT mapping macro.

```
PGSER L,OUT,LA=LOADWORD
IHAPVT
```

Example 6

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
```

PGSER — Fast Path Page Services

Description

The fast path PGSER macro performs FIX and FREE requests for users on performance paths.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and key 0
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 for I/O and external interrupts
Locks:	No spin locks can be held.
Control parameters:	Must be in the primary address space

Programming Requirements

The caller must include the IHAPVT mapping macro.

Restrictions

The following restrictions apply to the fast path services:

- Short term fixes only
- No ECB can be specified
- No TCB can be specified
- No VIO window page scan be specified
- When the list form of the macro is being used, all user-defined short page service lists (SSLs) must be valid in nonpageable storage.
- **IBM recommends that page fixes of more than 100 pages be divided into several smaller fix requests**. Large page fix requests can cause an excessive spin loop to occur.

The fast path PGSER macro does not verify any of the restricted conditions. The caller is responsible for verifying the restricted conditions and providing recovery to purge FIX requests when the task terminates before a page service request is complete.

Input Register Information

Before issuing the PGSER macro, the caller must ensure that GPR 13 points to a standard 18-word save area in nonpageable storage.

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-4	Used as work registers by the system
5-13	Unchanged
14-15	Used as work registers by the system

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

© Copyright IBM Corp. 1988, 2000

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 fast path 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

,FIX
,FREE

   ,LA=list addr           list addr: RX-type address or register (1), (5) - (12).
                           Note: This parameter is valid only if L is specified.

   ,A=start addr           start addr: RX-type address or register (1), (5) - (12).
                           Note: This parameter is valid only if R is specified.

   ,EA=ending addr         ending addr: RX-type address or register (2), (5) - (12).
                           Note: This parameter is valid only if R is specified.

   ,BACKOUT=Y              Default: BACKOUT=Y
   ,BACKOUT=N              Note: This parameter is valid only for FIX requests.

   ,ASCB=ascb addr         ascb addr: RX-type address or register (5) - (12).

   ,RELATED=value          value: Any valid macro keyword specification.

,BRANCH=SPECIAL
```

Parameters

The parameters are explained as follows:

R
L Specify the manner in which the input is supplied. If R is specified, the user supplies the starting and ending addresses of the virtual storage area for which the service is to be performed. If L is specified, the user supplies the address of the short page services list (SSL), which specifies the virtual storage area for which the service is to be performed. See the topic “Input to Page Services” in *OS/390 MVS Programming: Authorized Assembler Services Guide* for a description of the SSL.

,FIX
,FREE
Indicate the function to be performed.

FIX specifies that the virtual storage areas are to reside in central (also called real) storage and are ineligible for page-out while the address space is swapped in. This parameter does not prevent pages from being paged out when the entire address space is swapped out of central storage. FIX will ignore a request to fix storage in a system area that has the fixed attribute (for example, the LSQA and SQA). A FIX request for a page in the LSQA or SQA will not cause the page to be backed by central storage below 16 megabytes.

FREE specifies that the virtual storage areas that were previously fixed through the FIX option are eligible for page-out. A fixed page is not considered pageable until the number of FREE and FIX requests for the page are equal.

,LA=*list addr*
Specifies the address of the short page service list (SSL) for L requests.

,A=*start addr*
Specifies the address of the start of the virtual area for R requests.

,EA=*end addr*
Specifies the last byte on the last page of the virtual area for R requests.

,BACKOUT=Y
,BACKOUT=N
Specify the procedure to follow if an unallocated page is encountered during the processing of a fix request.

If BACKOUT=Y is specified, all pages fixed as part of the request will be freed before control returns to the caller.

If BACKOUT=N is specified, the pages previously fixed as part of the request will not be freed before control returns to the caller. In this situation, no further pages are processed once an unallocated page is encountered.

,ASCB=*ascb addr*
Specifies the address of the ASCB for the currently addressable address space.

Note: The ASCB must reside in 24-bit addressable storage.

,RELATED=*value*
Specifies information used to document the macro and to relate the service performed to some corresponding service or function. The format of the information specified can be any valid coding values that the user chooses.

,BRANCH=SPECIAL
Specifies a branch entry call to the fast path FIX and FREE services.

ABEND Codes

None.

Return and Reason Codes

None.

Example 1

Fix 4096 bytes of storage starting at the address BUFFER. The address of the ASCB is in register 6. Include the IHAPVT mapping macro.

```
PGSER R,FIX,A=BUFFER,EA=BUFFER+4095,BRANCH=SPECIAL,ASCB=(6)
IHAPVT
```

Example 2

Free the area specified in the SSL defined at LISTSSL. Use the ASCB in PSAAOLD. Include the IHAPVT mapping macro.

```
L   5,PSAAOLD
PGSER L,FREE,LA=LISTSSL,ASCB=(5),BRANCH=SPECIAL
IHAPVT
```

POST — Signal Event Completion

Description

Use the POST macro to set a specified 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.

The bits in the ECB are set as follows:

- Bit 0 of the specified ECB is set to 0 (wait bit).
- Bit 1 is set to 1 (complete bit).
- Bits 2 through 31 are set to the specified completion code.

The POST macro is also described in *OS/390 MVS Programming: Assembler Services Reference* with the exception of the parameters ASCB, ERRET, ECBKEY, and LINKAGE=BRANCH. For further information on how to use POST to serialize parallel tasks, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for callers of POST with LINKAGE=SVC or LINKAGE=SYSTEM are:

Minimum authorization: Problem state with any PSW key. For the ASCB, ERRET, and ECBKEY parameters, one or more of the following:
- Supervisor state
- PSW key 0-7
- APF-authorized

Dispatchable unit mode: Task or SRB. (Programs running in SRB mode may use only LINKAGE=SYSTEM or LINKAGE=BRANCH.)

Cross memory mode: One of the following:
- For LINKAGE=SVC: PASN=HASN=SASN
- For LINKAGE=SYSTEM: Any PASN, any HASN, any SASN

AMODE: 24- or 31-bit

ASC mode: Primary

Interrupt status: Enabled for I/O and external interrupts

Locks: No locks held

Control parameters: If the caller specifies the ASCB parameter, the event control block (ECB) must be addressable from the address space identified by the ASCB parameter. If the caller does not specify the ASCB parameter, the ECB must be in the primary address space.

The requirements for callers of POST with LINKAGE=BRANCH are:

Minimum authorization: Supervisor state and PSW key 0

Dispatchable unit mode: Task or SRB

Cross memory mode:
- If the caller specifies the ASCB parameter: PASN¬=HASN¬=SASN
- If the caller does not specify ASCB: PASN=HASN=SASN or PASN¬=HASN¬=SASN

AMODE: 24- or 31-bit

ASC mode: Primary or secondary

Interrupt status: Enabled or disabled for I/O and external interrupts

Locks: If the caller specifies the ASCB parameter, the caller may hold the local lock, but is not required to hold any locks. If the caller does not specify ASCB, the caller must hold the local lock.

Control parameters: If the caller specifies the ASCB parameter, the event control block (ECB) must be addressable from the address space identified by the ASCB parameter. If the caller does not specify the ASCB parameter, the ECB must be in the primary address space.

The requirements for callers of POST with LINKAGE=BRANCH,ECBKEY=*key* are:

© Copyright IBM Corp. 1988, 2000

Minimum authorization:	Supervisor state and PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	Local lock held
Control parameters:	If the caller specifies the ASCB parameter, the event control block (ECB) must be addressable from the address space identified by the ASCB parameter. If the caller does not specify the ASCB parameter, the ECB must be in the primary address space.

Programming Requirements

For LINKAGE=BRANCH or BRANCH=YES, the caller must include the CVT mapping macro.

Restrictions

Callers that specify LINKAGE=SVC cannot have any enabled unlocked task (EUT) functional recovery routines (FRR) established.

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.

Output Register Information

For LINKAGE=SVC, 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

For LINKAGE=SYSTEM, 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

For LINKAGE=BRANCH, when control returns to the caller the general purpose registers (GPRs) contain one of the following:

- If the ASCB parameter is not specified:

Register	Contents
0-9	Unchanged
10-11	Used as work registers by the system
12-13	Unchanged
14-15	Used as work registers by the system

- If the ASCB parameter is specified, the LOCAL lock is held and MEMREL=YES is specified (or defaulted):

Register	Contents
0	One of the following: – If the ECBKEY parameter *is not* specified: Unchanged – If the ECBKEY parameter *is* specified: Used as a work register by the system
1-9	Unchanged

10-15	Used as work registers by the system

- If the ASCB parameter is specified, the LOCAL lock is not held or MEMREL=NO is specified:

Register	Contents
0-8	Used as work registers by the system
9	Unchanged
10-15	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 POST macro is written as follows:

---

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

---

ecb addr	*ecb addr*: RX-type address, or register (2) - (12), except (10).
,*comp code*	*comp code*: Symbol, decimal or hexadecimal digit, or register (0), (2) - (9), (10), or (12). **Range of values**: 0 - 2^{30}-1, **Default**: 0
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,ERRET=*err addr*,ECBKEY=*key*	*addr*: RX-type address, or register (2) - (9), (12). *err addr*: RX-type address, or register (2) - (9), (12). *key*: Symbol, decimal or hexadecimal digit, or register (2) - (9), (12). **Range of values**: 0 - 15 (decimal), **Default**: None. **Note**: If the register form is specified, bits 24-27 of the register must contain the key.
,LINKAGE=SVC ,LINKAGE=BRANCH	**Default**: LINKAGE=SVC
,LINKAGE=BRANCH, ECBKEY=*key*	*key*: Symbol, decimal or hexadecimal digit, or register (2) - (9), (12). **Range of values**: 0 - 15 (decimal), **Default**: None.
,LINKAGE=SYSTEM	
,LINKAGE=SYSTEM, ERRET=*err addr*	*err addr*: RX-type address, or register (2) - (9), (12). **Note**: If the register form is specified, bits 24-27 of the register must contain the key.
,BRANCH=NO ,BRANCH=YES	**Default**: BRANCH=NO
,MEMREL=YES	**Default**: MEMREL=YES

,MEMREL=NO	**Note**: MEMREL can be coded only if LINKAGE=BRANCH and the ASCB and ERRET parameters are coded.
,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.

,ASCB=*addr***,ERRET=***err addr*
,ASCB=*addr***,ERRET=***err addr***,ECBKEY=***key*
Specifies the address of the ASCB of the address space containing the ECB being posted, and a pointer to the address of the routine that receives control when an error condition resulting from a POST failure is detected. The ASCB must reside in 24-bit addressable storage.

You can also specify the storage protection key of the ECB to be posted. The system checks the storage key of the ECB against the ECBKEY before posting it.

If there is a wait condition against the ECB, the system also ensures that the waiting task is in a valid key to modify the ECB. (See "Cross Memory Post" in *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information.)

,LINKAGE=SVC
,LINKAGE=BRANCH
,LINKAGE=BRANCH,ECBKEY=*key*
,LINKAGE=SYSTEM
,LINKAGE=SYSTEM,ERRET=*err addr*
Specifies the type of linkage from the caller to a system service routine that POST invokes. The default is LINKAGE=SVC.

For LINKAGE=SVC, the linkage is through an SVC instruction. This linkage is valid only when the caller is in task mode and primary ASC mode, where primary, home, and secondary are the same address space. For SVC callers, registers 2-14 are preserved.

For LINKAGE=BRANCH, the linkage is through a branch entry. This linkage is valid when the caller is in primary or secondary ASC mode. The calling requirements and the registers that are preserved depend on the other parameters specified, as follows:

- If ASCB is not specified, the caller must hold the local lock and be in noncross memory mode. Registers 0-9, 12, and 13 are preserved.
- If ASCB is specified, the MEMREL parameter and the LOCAL lock determine the calling requirements and registers saved.
  - If the LOCAL lock is held and MEMREL=YES is specified (or defaulted), then the current address space must be the home address space and registers 1-9 are preserved. If the ECBKEY parameter is not specified, register 0 is also preserved.
  - If the LOCAL lock is not held or MEMREL=NO is specified, then only register 9 is preserved. The current address space can be any address space.

With LINKAGE=BRANCH, you can also specify the storage protection key of the ECB to be posted using the ECBKEY parameter. The way the key specified on the ECBKEY parameter is processed depends on whether or not there is a wait condition against the ECB:

- If there is no wait condition against the ECB, the system checks the storage key of the ECB before posting it.

- If there is a wait condition against the ECB, the system ignores any value specified on the ECBKEY parameter. Instead, the system ensures that the waiting task is in a valid key to post the macro. (See “Cross Memory Post” in *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information.)

Note: BRANCH=YES and BRANCH=NO are still supported by the system, but LINKAGE is the recommended parameter.

For LINKAGE=SYSTEM, the linkage uses a non-SVC entry. Callers must be enabled, unlocked, and in primary ASC mode. This linkage is valid for callers in both noncross memory and cross memory mode. If you specify the ASCB parameter, the ECB must be addressable from the address space identified by ASCB. If you do not specify ASCB, the ECB must be in the caller's primary address space. When you specify LINKAGE=SYSTEM and ASCB, you must also specify ECBKEY.

- ERRET=*err_addr* specifies a pointer to the address of the routine that gets control when the system detects a POST failure. If the caller is not authorized, the error routine does not receive control. When you have specified LINKAGE=SYSTEM without ASCB=, ERRET is only needed when you are posting an extended ECB and the primary address space is different from the home address space.
- When you issue LINKAGE=SYSTEM, the POST macro service issues the return codes described in “Return Codes” on page 188.

  LINKAGE=SYSTEM without the ASCB parameter is intended to be used by programs in cross memory mode.

,MEMREL=YES
,MEMREL=NO

Specifies the address space in which the routine specified on the ERRET parameter is to run:

- Specify MEMREL=YES (or accept the default of MEMREL=YES) if you want the ERRET routine to run in the caller's home address space.
- Specify MEMREL=NO if you want the ERRET routine to run in the master scheduler's address space.

Note: You cannot specify MEMREL=YES if you hold the local lock and you are running in cross memory mode. Therefore, if you do not know that the primary and home address spaces are the same, you should specify MEMREL=NO.

,RELATED=*value*

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

ABEND Codes

POST might abnormally terminate with one of the following abend codes:

- X'102'
- X'202'
- X'302'
- X'402'
- X'502'
- X'602'
- X'702'

These hexadecimal codes are described in *OS/390 MVS System Codes*.

Return Codes

When you issue LINKAGE=SYSTEM, the POST macro service issues the following hexadecimal return codes.

Figure 37. Return Codes for the POST Macro

Return Code	Meaning and Action
00	**Meaning**: Indicates a synchronous POST was done, as requested. **Action**: None.
04	**Meaning**: Environmental error. Indicates an asynchronous POST is in progress. If you specified ERRET and a failure occurs before the POST completes, the error routine that you specified will receive control. If you did not specify ERRET and a failure occurs before the POST completes, no error routine exists to receive control. **Action**: None required. However, you might take some action based upon your application.
08	**Meaning**: Environmental error. Indicates an asynchronous POST is in progress. You specified ERRET; however, if an error occurs before POST completes, the error routine that you specified will not receive control. **Action**: None required. However, you might take some action based upon your application.

Example 1

Post an event control block whose address is ECB, where the address space containing the ECB has an ASCB specified by register 5, and where ERRRTN is the routine to be given control on error conditions.

```
POST ECB,ASCB=(REG5),ERRET=ERRRTN
```

Example 2

Post the ECB from example 1 with a hexadecimal completion code of 3FF.

```
POST ECB,X'3FF',ASCB=(REG5),ERRET=ERRRTN
```

Example 3

Post the ECB from example 1 using a stacking PC for linkage. The address of the error routine is in register 3.

```
POST ECB,LINKAGE=SYSTEM,ECBKEY=0,ASCB=(REG5),ERRET=(REG3)
```

POST—List Form

Syntax

The list form of 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*: A-type address.
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,ERRET=*err addr*,ECBKEY=YES	*addr*: A-type address.

	err addr: A-type address.
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

Parameters

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

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

POST—Execute Form

Syntax

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

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

ecb addr	*ecb addr*: RX-type address, or register (2) - (12).
,comp code	*comp code*: Symbol, decimal or hexadecimal digit, or register (0) or (2) - (12). **Range of values**: 0 - 2^{30}-1
,ASCB=*addr*,ERRET=*err addr* ,ASCB=*addr*,ERRET=*err addr*,ECBKEY=*key*	*addr*: RX-type address, or register (2) - (12). *err addr*: RX-type address, or register (2) - (12). *key*: Symbol, decimal or hexadecimal digit, or register (2) - (12). **Range of values**: 0 - 15 (decimal), **Default**: None. **Note**: If the register form is specified, bits 24-27 of the register must contain the key.
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=(E,*prob addr*)	*prob addr*: RX-type address, or register (1) or (2) - (12).

Parameters

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

,MF=(E,*prob addr*)
Specifies the execute form of the POST macro using a remote control program parameter list.

PTRACE — Processor Trace

Description

The PTRACE macro creates a trace table entry and places it in the system trace table. The entry consists of an event identifier, the contents of a designated range of general registers or storage locations, and system supplied status information.

When using this macro, the user must provide the following information:

- The type of trace entry that is to be created
- The data to be recorded in the trace entry

The PTRACE macro can only be issued with DAT-ON. The caller must be in key 0 and supervisor state but can be in cross memory mode and in either 24 or 31-bit addressing mode. All addresses passed to the PTRACE routine are treated as 31-bit addresses. PTRACE users must include the IHAPSA and IHATRVT mapping macros and register 13 must point to a 72-byte save area that can be used by the PTRACE service.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state, key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt Status:	Enabled or disabled for I/O or external interrupts
Locks:	Any locks may be held
Control parameters:	Must be in the primary address space

Programming Requirements

The PTRACE macro can only be issued with DAT-ON. All addresses passed to the PTRACE routine are treated as 31-bit addresses. PTRACE users must include the IHAPSA and IHATRVT mapping macros and register 13 must point to a 72-byte save area that can be used by the PTRACE service.

Restrictions

None.

Output Register Information

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

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

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

© Copyright IBM Corp. 1988, 2000

Performance Implications

None.

Syntax

The PTRACE macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede PTRACE.
PTRACE	
ƀ	One or more blanks must follow PTRACE.
TYPE=USR*n*	*n*: hexadecimal digit 0 - F.
,REGS=(reg1,reg2) ,REGS=(1)	**Default**: REGS=(1) reg1: decimal digit 2 - 12. reg2: decimal digit 2 - 12.
,SAVEAREA=STANDARD	

Parameters

The parameters are explained as follows:

TYPE=USR*n*
Specifies a user-event explicit trace entry. The hexadecimal number, *n*, identifies the entry. Trace processing places this number in the trace entry for identification purposes.

,REGS=(*reg1,reg2***)**
,REGS=(1)
Defines the data to be placed in the user's trace entries. Multiple trace entries are created if more than 5 registers or 5 words of data are requested.

If REGS=(*reg1,reg2*) is specified, the data is located in a range of registers, where *reg1* specifies the first register in the range and *reg2* specifies the last register in the range. The register number, *reg2*, must always be greater than or equal to the register number, *reg1*. A maximum of 11 words of data can be indicated for tracing using REGS=(*reg1,reg2*).

If REGS=(1) is specified or used as the default, register 1 must contain the 31-bit address of a parameter list. The high order bit of this address must be set to 0. If REGS=(1) is specified, up to 1024 words of data can be recorded. The parameter list contains N+1 fullword entries. The first word contains the number of words of data (N) to be recorded. This is followed by the N words of data to be placed in the user's trace entries.

,SAVEAREA=STANDARD
Specifies that register 13 contains the address of a 72-byte save area that can be used by the PTRACE routine.

When control returns to the caller, registers 2-13 are restored to their original values, but the original contents of registers 0, 1, 14, and 15 are destroyed. On exit, register 15 contains a return code.

Return and Reason Codes

The hexadecimal return code from the PTRACE macro is as follows:

Figure 38. Return Code for the PTRACE Macro

Return Code	Meaning
00	**Meaning**: The function completed successfully.

Example 1

Create a trace table entry for user event 4. Registers 5, 6, and 7 contain the user data to be recorded.

```
PTRACE TYPE=USR4,REGS=(5,7),SAVEAREA=STANDARD
```

Example 2

Create trace table entries for user event C. Register 1 contains the address of a parameter list containing the data to be recorded.

```
PTRACE TYPE=USRC,REGS=(1),SAVEAREA=STANDARD
```

PURGEDQ — Purge SRB Activity

Description

The PURGEDQ macro allows a task to purge particular SRB activity. The system dispatches an SRB routine asynchronously from when the SCHEDULE macro was issued. For this reason, the conditions that existed in the system at the time the SCHEDULE was issued might have changed by the time the routine is dispatched. If the environment that the asynchronous routine requires to run successfully has been changed, the results are unpredictable. For this reason, the PURGEDQ macro is available to:

- Dequeue SRBs not yet dispatched.
- Allow processing for dispatched SRBs to complete.
- “Clean up” each dequeued SRB.

The parameters on PURGEDQ determine the target address space and limit the scope of the purge. When purging SRBs scheduled in the primary address space, PURGEDQ waits for dispatched SRBs to finish. When purging SRBs scheduled in an address space other than the primary, PURGEDQ does not purge SRBs that have been dispatched, nor does PURGEDQ wait for dispatched SRBs to complete.

When the target address space is not the primary address space, PURGEDQ does not guarantee that all SRBs matching the purge parameters will be purged. The issuer of PURGEDQ is not informed of SRBs that are not purged. When purging SRBs scheduled in an address space other than primary, use a resource manager termination routine (RMTR) if you need to know whether a particular SRB has been purged.

Except for the TCB, all input parameters to this macro can reside in storage above 16 megabytes if the issuer is executing in 31-bit addressing mode.

See *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information on using the PURGEDQ macro, especially the resource manager termination routine (RMTR).

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state or PSW key 0 - 7 or APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=SASN=HASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the caller's primary address space

Programming Requirements

None.

Restrictions

None.

© Copyright IBM Corp. 1988, 2000

Input Register Information

Before issuing the PURGEDQ 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

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

When 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

Performance Implications

None.

Syntax

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

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

RMTR=*RMTR addr*	*RMTR addr*: RX-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr*: RX-type address, or register (2) - (12).
,ASIDTCB=*addr*	*addr*: RX-type address of an 8-byte field, or register (2) - (12) that contains an address of an 8-byte field.

Parameters

The parameters are explained as follows:

RMTR=*RMTR addr*

Specifies the address of the RMTR. If the program scheduled the SRB using SCHEDULE, this is the address that was placed in the SRBRMTR field (mapped by IHASRB). If the program scheduled the SRB using IEAMSCHD, this is the address

specified on the RMTRADDR parameter. It limits the purge to SRBs that are protected by the same RMTR; that is, where the SRBs have the same address.

,ASID=*ASID addr*
Specifies the address of a halfword that contains the ASID of the target address space into which the SRB was scheduled. If you omit ASID, the system assumes that the primary address space is the target address space. Note that when you use the ASID parameter to purge SRBs scheduled to an address space other than primary, PURGEDQ does not guarantee that all SRBs will be purged.

,ASIDTCB=*addr*
Specifies the address of a doubleword that describes the TCB for which SRBs are to be purged. Through this parameter, you can purge the SRBs associated with a specific task. If you omit the parameter, the system purges SRBs associated with the current task in the primary address space.

When you use the ASID parameter to purge SRBs scheduled to an address space other than primary, PURGEDQ does not guarantee that all SRBs will be purged.

Specify the ASIDTCB parameter in one of the following ways:

1. To attempt to purge all SRBs scheduled to a specific address space as defined by ASID, set the ASIDTCB parameter as follows:

Set These Bytes	To This Value	Meaning
Bytes 0-7	Zero	The system is to purge all SRBs defined by the ASID (SRBASCB) and RMTR parameters, regardless of their task (SRBPTCB) and address space (SRBPASID) association.

2. To purge all SRBs scheduled by a specified address space, set the ASIDTCB parameter as described below:

Set These Bytes	To This Value	Meaning
Bytes 0-1 Bytes 2-3 Bytes 4-7	Reserved ASID1 Zero	The system is to purge all SRBs defined by the ASID and RMTR parameters associated with the target address space (SRBPASID), regardless of their task (SRBPTCB).

3. To purge all SRBs associated with a specified TCB in a specified address space, set the ASIDTCB parameter as described below:

Set These Bytes	To This Value	Meaning
Bytes 0-1 Bytes 2-3 Bytes 4-7	Zero ASID2 TCB address	The system is to purge all SRBs defined by the ASID and RMTR parameters associated with the scheduling address space (SRBPASID) and task (SRBPTCB). (If you specify SRBPTCB, you must also specify SRBPASID.)

Note: The TCB resides in storage below 16 megabytes.

ABEND Codes

17B
27B
47B

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

Example 1

Purge all SRBs scheduled to ASID '20'X with:

- SRBPTCB equal to the current TCB (that is, the TCB issuing the PURGEDQ)
- SRBPASID equal to the primary ASID
- SRBRMTR equal to the address of RMTR routine RMTRA

```
         PURGEDQ ASID=AS1,RMTR=RMTRA

AS1      DC   XL2'0020'
```

Example 2

Purge all SRBs scheduled to ASID '21'X, regardless of what is specified in SRBPASID and SRBPTCB, and that have SRBRMTR equal to the address of RMTR routine RMTRB.

```
         PURGEDQ ASID=AS2,ASIDTCB=PURGPRM1,RMTR=RMTRB

PURGPRM1 DC    XL8'00000000'
AS2      DC    XL2'0021'
```

Example 3

Purge all SRBs scheduled to the primary address space (that is, the address space from which this PURGEDQ was issued) that have:

- SRBPASID of '12'X
- SRBPTCB equal to the address of TCBX
- SRBRMTR equal to the address of RMTR routine RMTRC

```
         PURGEDQ ASIDTCB=PURGPRM2,RMTR=RMTRC

PURGPRM2 DS    0CL8
         DC    XL2'0000'
PURGASID DC    XL2'0012'
PURGTCB  DC    A(TCBX)
```

Example 4

Purge all SRBs scheduled into the primary address space, related to the current (terminating) task, and associated with the resource manager termination routine located at RESCLEAN.

```
PURGEDQ RMTR=RESCLEAN
```

PURGEDQ—List Form

For programs that require reentrant code, use the list form of the PURGEDQ macro together with the execute form of the macro. The list form of the macro defines an area of storage that the execute form of the macro uses to store parameter values.

Syntax

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

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

```
PURGEDQ
```

ъ	One or more blanks must follow PURGEDQ.
RMTR=*RMTR addr*	*RMTR addr*: A-type address.
,ASID=*ASID addr*	*ASID addr*: A-type address.
,ASIDTCB=*addr*	*addr*: A-type address.
,MF=L	

Parameters

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

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

Example

Specify the resource manager termination routine located at RESCLEAN and produce the parameter list to be used by the execute form of the PURGEDQ macro.

```
STATPDQ PURGEDQ RMTR=RESCLEAN,MF=L
```

PURGEDQ—Execute Form

For programs that require reentrant code, use the execute form of the PURGEDQ macro together with the list form. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

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

name	*name*: Symbol. Begin *name* in column 1.
ъ	One or more blanks must precede PURGEDQ.
PURGEDQ	
ъ	One or more blanks must follow PURGEDQ.
RMTR=*RMTR addr*	*RMTR addr*: RX-type address, or register (2) - (12).
,ASID=*ASID addr*	*ASID addr*: RX-type address, or register (2) - (12).
,ASIDTCB=*addr*	*addr*: RX-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 PURGEDQ macro, with the following exception:

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

Example

Purge all SRBs scheduled into the address space given in register 6 and associated with the resource manager termination routine located at RESCLEAN. Indicate that the remote control program parameter list is located at STATPDQ.

```
PURGEDQ ASID=(6),RMTR=RESCLEAN,MF=(E,STATPDQ)
```

QEDIT — Command Input Buffer Manipulation

Description

The QEDIT macro generates the required entry parameters and processes the command input buffer for the following uses:

- Dechaining and freeing of a command input buffer (CIB) from the CIB chain for a task.
- Setting a limit for the number of CIBs that may be simultaneously chained for a task.

Syntax

The QEDIT macro is written as follows:

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede QEDIT.
QEDIT	
␣	One or more blanks must follow QEDIT.
ORIGIN=*CIB addr ptr*	*CIB addr ptr*: RX-type address, or register (0),(2) - (12).
,BLOCK=*CIB addr* ,CIBCTR=*CIB nmbr*	*CIB addr*: RX-type address, or register (1), (2) - (12). *CIB nmbr*: Decimal digit, with a maximum value of 255 or register (1), (2) - (12).

Parameters

The parameters are explained as follows:

ORIGIN=*CIB addr ptr*
Specifies the address of the pointer to the first CIB chain for the task. This address is obtained using the EXTRACT macro. If BLOCK and CIBCTR are omitted, the caller must be executing under PSW key 0-7; in this case, the entire CIB chain is freed. The system prevents problem state programs from freeing the entire CIB chain.

,BLOCK=*CIB addr*
Specifies the address of the CIB to be freed from the CIB chain for a task.

,CIBCTR=*CIB nmbr*
Specifies the limit for the number of CIBs to be chained at any time for a task.

Notes:

1. When using any address returned from the EXTRACT macro as input to the QEDIT macro, the user must use the IEZCOM mapping macro to establish addressability based on the address returned by EXTRACT.
2. The CIB must reside in 24-bit addressable storage.

© Copyright IBM Corp. 1988, 2000

Return and Reason Codes

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

Figure 39. Return Codes for the QEDIT Macro

Return Code	Meaning
00	**Meaning**: The required function was successfully completed.
04	**Meaning**: The CIB to be deleted was not found on any CIB chain.
08	**Meaning**: The limit for the number of CIBs to be chained was exceeded; an issuer who made a request to free all the CIBs on a chain was not in supervisor state and PSW key zero; or the user provided an invalid address for the pointer to the CIB chain, an invalid address for the CIB address, or an invalid CIB number as input to the macro.

Example 1

Free the entire CIB chain, where register 8 contains the address of the pointer to the CIB chain.

```
QEDIT ORIGIN=(8)
```

Example 2

Free the CIB whose address is in register 5 from the CIB chain. Register 8 contains the address of the pointer to the CIB chain.

```
QEDIT ORIGIN=(8),BLOCK=(5)
```

RACF Macros

See *OS/390 SecureWay Security Server External Security Interface (RACROUTE) Macro Reference* for the descriptions of the following macros:

- FRACHECK
- RACDEF
- RACHECK
- RACINIT
- RACLIST
- RACROUTE
- RACSTAT
- RACXTRT

© Copyright IBM Corp. 1988, 2000

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 “Finding the UCB Address for the RESERVE Macro” in *OS/390 MVS Programming: Authorized Assembler Services Guide*.

If global resource serialization is active, the hardware RESERVE can be suppressed leaving a SYSTEMS ENQ depending on the contents of the resource name lists. See *OS/390 MVS Planning: Global Resource Serialization* for information on resource name lists.

A RESERVE used with the MASID and MTCB operands provides a special form of the RESERVE macro that allows a further conditional control of a resource. One task, called the “issuing task” can issue a RESERVE macro for a resource specifying the ASID and TCB of another task, called the “matching task”.

The RESERVE macro is also described in *OS/390 MVS Programming: Assembler Services Reference*, with the exception of the MASID, MTCB, and ECB parameters.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with any PSW key. For the MASID, MTCB, and ECB parameters, **one of the following**:

- Supervisor state
- PSW key 0-7
- APF-authorized.

Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 24- or 31-bit
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: If the caller's AMODE is 24-bit, all parameters must reside below 16 megabytes.

Programming Requirements

Before issuing the RESERVE macro with a UCB address, an authorized caller must serialize the UCB against dynamic I/O reconfiguration requests. The caller can accomplish this serialization by allocating or pinning the UCB. Such serialization ensures that a dynamic I/O reconfiguration request does not delete or reuse the UCB before the RESERVE macro uses the address.

© Copyright IBM Corp. 1988, 2000

Restrictions

If a task issues two RESERVE macros for the same device without an intervening DEQ macro, the task ends abnormally unless the second RESERVE specifies the keyword parameter RET or ECB. (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 *OS/390 MVS Programming: Assembler Services Guide*. For further information about limiting global resource serialization requests, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

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, RET=HAVE, or the ECB parameter: • 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.
ъ	One or more blanks must precede RESERVE.

RESERVE

ъ	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	
,ECB=*ecb addr*	*ecb addr*: A-type address, or register (2) - (12).
,UCB=*ucb addr*	*ucb addr*: A-type address, or register (2) - (12).
,LOC=BELOW	**Default:** LOC=BELOW
,LOC=ANY	
,MASID=*matching-asid addr*	*matching-asid addr*: A-type address, or register (2) - (12).
,MTCB=*matching-tcb addr*	*matching-tcb addr*: A-type address, or register (2) - (12).
,RELATED=*value*	*value*: any valid macro keyword specification.

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.

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

,UCB=*ucb addr*
Specifies the address of a fullword that contains the address of the UCB for the device to be reserved. The UCB does not need to 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.

,MASID=*matching-asid addr*
Specifies the matching task (by defining a matching ASID) for the RESERVE. MASID defines the ASID of a task that may be using a resource desired by the issuer of the RESERVE macro.

Note: MASID can be specified only if MTCB is also specified.

,MTCB=*matching-tcb addr*
Specifies the matching task (by defining a matching TCB) for the RESERVE. MTCB defines the TCB of a task that may be using a resource desired by the issuer of the RESERVE macro.

Note: MTCB can be specified only if MASID is also specified.

If the task specified by the MASID and MTCB parameters is not using the resource, global resource serialization gives control to the issuer of the RESERVE and returns a return code indicating whether the resource can be used. If the task specified by MASID and MTCB parameters is using the resource, global resource serialization

records a request for the resource, suspends the issuing task until the resource is available, or optionally returns a return code indicating that an ECB will be posted when the resource can be used.

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

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

,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.

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 *OS/390 MVS System Codes* for explanations and responses for these codes.

Return and Reason Codes

Return codes are provided by the system only if you specify RET=TEST, RET=USE, RET=HAVE, or ECB; 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 40.

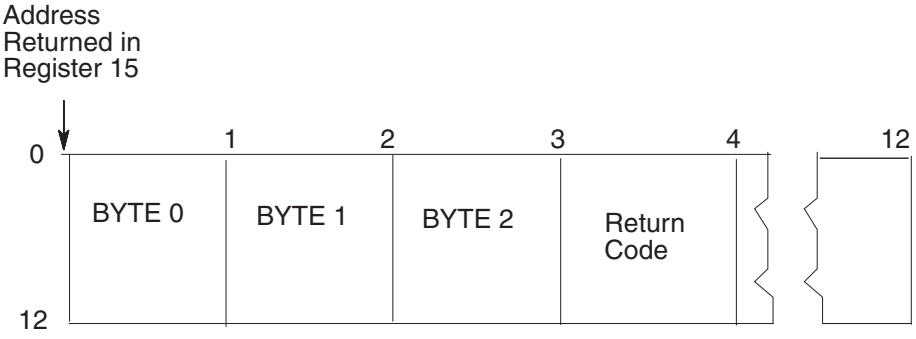


Figure 40. Return Code Area Used by RESERVE

The return codes for the RESERVE macro with the RET=TEST parameter are described in Figure 41.

Figure 41. 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.
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 40. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.
20	**Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource. **Action**: None required. However, you might take some action based on your application.

The return codes for the RESERVE macro with the RET=USE parameter are described in Figure 42.

Figure 42. 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 40 on page 209. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.
18	**Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. **Action**: Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.

The return codes for the RESERVE macro with the RET=HAVE parameter are described in Figure 43 on page 211.

Figure 43. 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 40 on page 209. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.
18	**Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. **Action**: Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.
20	**Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource. **Action**: The caller can use the resource, but it must ensure that the owning task does not terminate while the caller is using the resource. If the caller requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the caller requested shared control and the owning task requested shared control, this return code might indicate that a previous task had requested exclusive control. The caller must issue a DEQ macro to cancel this RESERVE request.
28	**Meaning**: The caller cannot obtain exclusive control of the resource using the RESERVE macro with the MASID and MTCB parameters. The matching task's involvement with other tasks precludes control by the caller. **Action**: This task must not issue a DEQ macro to cancel the RESERVE request.
44	**Meaning**: The caller is violating a restriction of using the RESERVE macro with the MASID and MTCB parameters in one or more of the following ways: • Another task has already issued the RESERVE macro for this resource specifying the same values for the MASID and MTCB parameters • The MASID and MTCB parameters specify a task that acquired control of the resource by using the RESERVE macro with the MASID and MTCB parameters • The matching task requested ownership of the resource but has not yet been granted ownership. **Action**: Do not use the resource; the caller does not have control of it.

The return codes for the RESERVE macro with the ECB parameter are described in Figure 44.

Figure 44 (Page 1 of 2). Return Codes for the RESERVE Macro with the ECB Parameter

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: The active task now has control of the resource. **Action**: Do not wait on the ECB; it will not be posted.
4	**Meaning**: The active task does not have control of the resource yet. The ECB will be posted when the system assigns control to that task. **Action**: Wait on the ECB if your program cannot continue processing without control of the resource.

Figure 44 (Page 2 of 2). Return Codes for the RESERVE Macro with the ECB Parameter

Hexadecimal Return Code	Meaning and Action
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: Do not wait on the ECB; it will not be posted. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of Byte 0 as shown in Figure 40 on page 209. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: Do not wait on the ECB; it will not be posted.
18	**Meaning**: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource. **Action**: Do not wait on the ECB; it will not be posted. Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.
20	**Meaning**: The matching task (the task specified in the MASID and MTCB parameters) owns the resource. **Action**: Do not wait on the ECB; it will not be posted. The caller can use the resource, but it must ensure that the owning task does not terminate while the caller is using the resource. If the caller requested exclusive control, then this return code indicates that the matching task is the only task that currently owns the resource. If the caller requested shared control and the owning task requested shared control, this return code might indicate that a previous task had requested exclusive control. The caller must issue a DEQ macro to cancel this RESERVE request.
24	**Meaning**: The caller that specifies the RESERVE macro with the MASID and MTCB parameters will have exclusive control after the ECB is posted. **Action**: Wait on the ECB. Once the ECB is posted, the caller may use the resource, but must ensure that the matching task does not terminate while the caller is using the resource. The caller must issue a DEQ macro to cancel the RESERVE request.
28	**Meaning**: The caller cannot obtain exclusive control of the resource using the RESERVE macro with the MASID and MTCB parameters. The matching task's involvement with other tasks precludes control by the caller. **Action**: Do not wait on the ECB; it will not be posted. The caller must not issue a DEQ macro to cancel the RESERVE request.
44	**Meaning**: The caller is violating a restriction of using the RESERVE macro with the MASID and MTCB parameters in one or more of the following ways: • Another task has already issued the RESERVE macro for this resource specifying the same values for the MASID and MTCB parameters • The MASID and MTCB parameters specify a task that acquired control of the resource by using the RESERVE macro with the MASID and MTCB parameters • The matching task requested ownership of the resource but has not yet been granted ownership. **Action**: Do not wait on the ECB; it will not be posted. Do not use the resource; the caller does not have control of it.

Example

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: A-type address.
   ,rname addr

   ,
   ,E
   ,S

   ,                       rname length: symbol or decimal digit.
   ,rname length

   ,
   ,SYSTEMS
)

   ,RET=TEST
   ,RET=USE
   ,RET=HAVE
   ,RET=NONE

   ,ECB=ecb addr           ecb addr: A-type address.

,UCB=ucb addr              ucb addr: A-type address or 0.

   ,LOC=BELOW              Default: LOC=BELOW
   ,LOC=ANY

   ,MASID=0
   ,MTCB=0

   ,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.

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

Note: If the ECB parameter is specified on the execute form of the macro, it also must be specified on the list form of the macro. If MASID and MTCB also are specified on the execute form, MASID=0 and MTCB=0 must be specified in the list form.

RESERVE—Execute Form

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

name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede RESERVE.
RESERVE	
␣	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	
,ECB=*ecb addr*	*ecb addr*: RX-type address, or register (2) - (12).
,UCB=*ucb addr*	*ucb addr*: RX-type address, or register (2) - (12).
,LOC=BELOW ,LOC=ANY	**Default:** LOC=BELOW

,MASID=*matching-asid addr* ,MTCB=*matching-tcb addr*	*matching-asid addr*: A-type address, or register (2) - (12). *matching-tcb addr*: A-type address, or register (2) - (12).
,RELATED=*value*	*value*: any valid macro keyword specification.
,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.

Note: If the ECB parameter is specified on the execute form of the macro, it also must be specified on the list form of the macro. If MASID and MTCB also are specified on the execute form, MASID=0 and MTCB=0 must be specified in the list form.

RESMGR — Add or Delete a Resource Manager

Description

RESMGR allows an authorized program to add (ADD parameter) or delete (DELETE parameter) a resource manager routine.

Upon completion of RESMGR ADD, the resource manager is established for a task or address space. On the TYPE parameter, you choose whether the resource manager routine receives control when a task (TYPE=TASK) or an address space (TYPE=ADDRSPC) terminates. On the ROUTINE parameter, you designate the routine and choose the kind of linkage the routine has with RTM:

- LINK macro
- Branch instruction
- PC instruction.

For information about the uses of resource managers, see *OS/390 MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state or PKM allowing key 0 - 7
Dispatchable unit mode:	Task or SRB. However, you cannot issue TCB=CURRENT in SRB mode.
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:	The caller can hold the local lock, depending on whether the caller is in noncross memory mode or cross memory mode: • In noncross memory mode, the caller can hold the local lock for the home (that is, primary) address space. • In cross memory mode, the caller can hold the local lock for the primary address space but not for the home address space. If the caller holds a local lock, it can also hold the CMS lock.
Control parameters:	Must be in the primary address space

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the RESMGR 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, 2000

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

The LINK option on the ROUTINE parameter might degrade the performance of the system during task and address space termination.

If you specify TCB=ALL and ASID=ALL, the system invokes the resource manager program for every task termination initiated by the system. You can improve system performance by specifying a particular task or ASID.

Syntax

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

---

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

ADD
DELETE

,TOKEN=*tokaddr*	*tokaddr*: A-type address or register (2) - (12).
,TYPE=ADDRSPC ,TYPE=TASK	
,ASID=CURRENT ,ASID=ALL ,ASID=*asid*	*asid*: A constant or register (2) - (12).
,TCB=CURRENT ,TCB=ALL ,TCB=*tcbaddr* ,TTOKEN=*ttoken*	 *tcbaddr*: A-type address or register (2) - (12). *ttoken*: A-type address or register (2) - (12).
,ROUTINE=(LINK, *pgname*) ,ROUTINE=(BRANCH, *pgaddr*) ,ROUTINE=(PC, *pcnum*)	*pgname*: C-type constant, A-type address, or register (2) - (12). *pgaddr*: A-type address or register (2) - (12). *pcnum*: A constant or register (2) - (12).

,ECB=*ecbaddr*	*ecbaddr*: A-type address or register (2) - (12).
,PARAM=*paddr*	*paddr*: A-type address or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

ADD
DELETE
Specifies whether a resource manager is to be added or deleted. You must specify the same values for TYPE, TCB, TTOKEN, and ASID on DELETE as you specified on those parameters for ADD. On DELETE, you must specify the token that ADD returned so the system can identify the resource manager that you want to delete.

Note that you can use RESMGR to delete a resource manager from the resource manager routine itself.

,TOKEN=*tokaddr*
Specifies the address of the fullword where you want the system to store the token that it returns after an ADD. The token represents the resource manager that the system added. On DELETE, however, you store the token in this fullword before invoking the delete function. TOKEN is required for both ADD and DELETE.

,TYPE=ADDRSPC
,TYPE=TASK
Specifies whether the resource manager is an address space resource manager (ADDRSPC) or a task resource manager (TASK). The default is address space. If you specify TYPE=ADDRSPC, you cannot specify TTOKEN=*ttoken*.

,ASID=CURRENT
,ASID=ALL
,ASID=*asid*
Specifies the ID of the address space or spaces to be monitored for termination (TYPE=ADDRSPCE) or the home address space or the primary address space (TYPE=TASK). If you want to monitor:

- The home address space, specify ASID=CURRENT
- All address spaces, specify ASID=ALL
- A specific address space, specify ASID=*asid.*

If TYPE=TASK, *asid* must be the home or primary address space.

,TCB=CURRENT
,TCB=ALL
,TCB=*tcbaddr*
,TTOKEN=*ttoken*
Specifies the task that the system is to monitor for termination. If you want to monitor:

- The current task, specify TCB=CURRENT. Note that a program in SRB mode or in cross memory mode cannot issue TCB=CURRENT.

  If your program is in cross memory mode and you want to monitor a task in the primary address space, do not specify TCB=CURRENT. In this case, specify the primary address space through ASID=*asid* and the task through TCB=*tcbaddr* or TTOKEN=*ttoken*.
- All tasks in the specified address space, specify TCB=ALL. If you specify TCB=ALL with ASID=ALL, the system monitors all tasks in all address spaces.
- A task in the primary or home address space, specify TCB=*tcbaddr* or TTOKEN=*ttoken*. Note that TTOKEN is not valid on TYPE=ADDRSPC.

If your program is in cross memory mode and it holds the local lock for the primary address space, it cannot request monitoring of a task in the home address space.

,ROUTINE=(LINK, *pgname***)**
,ROUTINE=(BRANCH, *pgaddr***)**
,ROUTINE=(PC, *pcnum***)**
Specifies:

- the type of linkage to be used by the system when giving control to the resource manager
- the resource manager to receive control when the task or address space terminates.

The section on resource managers in *OS/390 MVS Programming: Authorized Assembler Services Guide* describes the registers on entry, the resource manager parameter list (RMPL), and some of the responsibilities of the resource manager.

If you specify LINK, the resource manager receives control in the addressing mode of the module being linked to through the LINK macro. *pgname* is one of the following:

- A character constant of up to 8 characters.
- The address of 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.

If you specify PC, the resource manager receives control through a PC instruction. *pcnum* is the PC number of the PC instruction that gives control to the resource manager. The address space from which the resource manager is called must have the authority to issue the PC.

If you specify BRANCH, the resource manager receives control in 31-bit addressing mode and in primary ASC mode through a branch instruction, and the resource manager (whether address space termination resource manager or task termination resource manager) must reside in storage addressable from all address spaces in which it can get control. Note that for address space termination, resource managers run in master's address space (ASID 1). *pgaddr* is the address of the resource manager.

ROUTINE is required on the ADD request.

,ECB=*ecbaddr*
The processing to delete a resource manager might not be complete when RESMGR returns. If you require notification after DELETE has completed, code ECB. The ECB will be posted when DELETE is completed. Note, however, that DELETE may already be complete upon return, in which case the system does not post any completion ECB. Check the return code from RESMGR before you wait on the ECB.

The system associates the completion ECB with the home address space of the DELETE requestor. ECB is valid only for DELETE. You must specify either of the following when you specify ECB:

- TYPE=ADDRSPC and ASID=ALL
- TYPE=TASK and ASID=ALL and TCB=ALL.

,PARAM=*paddr*
Specifies the address of an 8-byte field containing parameter data to be used by the resource manager when it receives control. The parameter data must reside in the caller's primary address space. PARAM is valid only with ADD. A copy of the 8-byte field is passed to the resource manager as the second parameter.

,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 from the ADD Function

Return codes from the ADD function follow. A return code greater than 4 indicates that RESMGR did not establish a resource manager.

Figure 45 (Page 1 of 2). Return Codes from the ADD Function

Decimal Return Code	Meaning and Action
0	**Meaning**: The resource manager was successfully established. The word provided by the TOKEN parameter contains the token required to delete the resource manager. **Action**: None.
12	**Meaning**: Program error. The caller did not provide the address of a word to contain the token of the new resource manager. **Action**: Issue the macro again with the TOKEN parameter.
16	**Meaning**: Program error. The caller did not provide the resource manager description through the ROUTINE parameter. **Action**: Issue the macro again with the ROUTINE parameter.
20	**Meaning**: Program error. The TCB address provided did not represent a valid TCB. **Action**: Issue the macro again and ensure that the TCB address represents a valid TCB.
24	**Meaning**: Program error. The ASID provided did not represent a valid ASCB. **Action**: Issue the macro again and ensure that the ASCB address represents a valid ASCB.
28	**Meaning**: Program error. The request type was not ADD or DELETE. **Action**: Ensure that the parameter list created by the RESMGR macro is not inadvertently overlaid or the contents lost due to an assembler coding error.
32	**Meaning**: Environmental error. RESMGR was unable to obtain storage for a work area it needed to process the request. **Action**: Rerun your program one or more times. If the problem persists, check with the operator to see if another user in the installation is causing the problem, or if the entire installation is experiencing storage constraint problems.
36	**Meaning**: Program error. The caller held an incorrect lock. **Action**: Ensure only the allowable locks are held before the macro is issued.
40	**Meaning**: Program error. It is not valid to establish a task resource manager for a specific task that is not in the home or primary address space of the requestor. **Action**: Issue the RESMGR macro for a task within the home or primary address space.
44	**Meaning**: System error. An unrecoverable error occurred while processing the request. **Action**: Rerun your program one or more times. If the problem persists, record the return code and supply it to the appropriate IBM support personnel.
48	**Meaning**: Environmental error. RESMGR was unable to obtain storage to maintain information about RESMGR. **Action**: Rerun your program one or more times. If the problem persists, check with the operator to see if another user in the installation is causing the problem, or if the entire installation is experiencing storage constraint problems.

Figure 45 (Page 2 of 2). Return Codes from the ADD Function	
Decimal Return Code	**Meaning and Action**
52	**Meaning**: Program error. The caller is not authorized to use RESMGR. **Action**: Ensure that your program has the proper authorization.
56	**Meaning**: Environmental error. The TCB was already terminating and no more dynamic resource managers can be established for it. **Action**: Attempt to establish the resource manager before the task starts to terminate.
60	**Meaning**: Environmental error. The ASCB was already in termination and no more dynamic resource managers can be established for it. **Action**: Attempt to establish the resource manager before the address space starts to terminate.
64	**Meaning**: Program error. The TTOKEN parameter specified a task in an address space other than the home address space. **Action**: Issue the RESMGR macro from within the address space of the task represented by the specified TTOKEN.

Return Codes from the DELETE Function

Return codes from the DELETE function follow. A return code greater than 8 indicates that RESMGR did not delete a resource manager.

Figure 46 (Page 1 of 2). Return Codes from the DELETE Function	
Decimal Return Code	**Meaning and Action**
0	**Meaning**: The resource manager was successfully deleted. An ECB is never posted for this return code. **Action**: None.
4	**Meaning**: The resource manager is currently in use. It has been queued for deletion. The ECB, if provided, will be posted when the delete process has completed. **Action**: None.
8	**Meaning**: The resource manager was queued for deletion by a previous request. It is still active and will be deleted as soon as it is no longer in use. **Action** None.
12	**Meaning**: Program error. The caller did not provide a token to RESMGR. **Action**: Specify the TOKEN parameter and value that represents the resource manager to be deleted.
16	**Meaning**: Program error. The token provided did not represent a currently established resource manager. **Action**: Ensure that the token was properly saved after the resource manager was established. Also ensure that the resource manager had not previously been deleted either directly through RESMGR or indirectly through task or address space termination.
20	**Meaning**: Program error. The TCB address provided did not represent a valid TCB. **Action**: Ensure that the TCB represents a valid TCB within the home address space.
24	**Meaning**: Program error. The ASID provided did not represent a valid ASCB. **Action**: Ensure that the ASCB address represents a valid ASCB.
28	**Meaning**: Program error. The request type was not ADD or DELETE. **Action**: Ensure that the parameter list created by the RESMGR macro was not inadvertently overlaid or its contents lost due to an assembler coding error.

Figure 46 (Page 2 of 2). Return Codes from the DELETE Function

Decimal Return Code	Meaning and Action
32	**Meaning**: Environmental error. RESMGR was unable to obtain storage for a work area it needed to process the request. **Action**: Rerun your program one or more times. If the problem persists, check with the operator to see if another user in the installation is causing the problem, or if the entire installation is experiencing storage constraint problems.
36	**Meaning**: Program error. The caller held an incorrect lock. **Action**: Ensure only the allowable system locks are held prior before the macro is issued.
40	**Meaning**: Program error. It is not valid to delete a task resource manager for a specific task that is not in the home or primary address space of the requestor. **Action**: Issue the RESMGR macro again for a task within the home or primary address space.
44	**Meaning**: System error. An unrecoverable error occurred while processing the request. **Action**: Rerun your program one or more times. If the problem persists, record the return code and supply it to the appropriate IBM support personnel.
48	**Meaning**: Program error. The ECB parameter was specified but is not supported for the particular type of delete request. **Action**: Refer to the ECB parameter description to ensure proper usage of this parameter.
52	**Meaning**: The caller is not authorized to use RESMGR. **Action**: Ensure that your program has the proper authorization.
64	**Meaning**: Program error. The TTOKEN parameter specified a task in an address space other than the home address space. **Action**: Issue the RESMGR macro from within the address space of the task represented by the specified TTOKEN.

Example 1

Establish a resource manager that receives control for every address space termination and every task termination. This resource manager is equivalent to having included the name IAMARESM in the IEAVTRML table.

```
RESMGR ADD,TOKEN=MYTOKEN,TYPE=ADDRSPC,ASID=ALL,
         ROUTINE=(LINK,'IAMARESM')

RESMGR ADD,TOKEN=MYTOKEN,TYPE=TASK,ASID=ALL,TCB=ALL,
         ROUTINE=(LINK,'IAMARESM')
```

Example 2

Establish a resource manager for the current task, using a branch interface and specifying the routine using register notation:

```
      L     R2,RMADDR    Obtain address of resource manager routine
      RESMGR ADD,TOKEN=MYTOKEN,TYPE=TASK,ASID=ALL,
               TCB=ALL,ROUTINE=(BRANCH,(R2))

      EXTRN RMROUTIN
RMADDR DC   A(RMROUTIN)  Address of resource manager routine
```

RESMGR—List Form

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

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

ADD DELETE	
,TOKEN=*tokaddr*	*tokaddr*: A-type address or register (2) - (12).
,TYPE=ADDRSPC ,TYPE=TASK	
,ASID=CURRENT ,ASID=ALL ,ASID=*asid*	*asid*: A constant.
,TCB=CURRENT ,TCB=ALL ,TCB=*tcbaddr* ,TTOKEN=*ttoken*	 *tcbaddr*: A-type address *ttoken*: A-type address
,ROUTINE=(LINK, *pgname*) ,ROUTINE=(BRANCH, *pgaddr*) ,ROUTINE=(PC, *pcnum*)	*pgname*: C-type constant or A-type address. *pgaddr*: A-type address. *pcnum*: A constant.
,ECB=*ecbaddr*	*ecbaddr*: A-type address.
,PARAM=*paddr*	*paddr*: A-type address.
,RELATED=*value*	*value*: Any valid macro keyword specification.
,MF=L	

Parameters

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

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

RESMGR—Execute Form

Use the execute form of RESMGR 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. You do not have to specify any parameters except MF on the execute form. For the parameters you do not specify on the execute form, RESMGR uses the parameters on the list form or their defaults.

Syntax

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

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

MF=(E,*listaddr*)	*listaddr*: RX-type address or register (2) - (12).
,ADD ,DELETE	
,TOKEN=*tokaddr*	*tokaddr*: A-type address or register (2) - (12).
,TYPE=ADDRSPC ,TYPE=TASK	
,ASID=CURRENT ,ASID=ALL ,ASID=*asid*	*asid*: A constant or register (2) - (12).
,TCB=CURRENT ,TCB=ALL ,TCB=*tcbaddr* ,TTOKEN=*ttoken*	 *tcbaddr*: RX-type address or register (2) - (12). *ttoken*: RX-type address or register (2) - (12).
,ROUTINE=(LINK, *pgname*) ,ROUTINE=(BRANCH, *pgaddr*) ,ROUTINE=(PC, *pcnum*)	*pgname*: a C-type constant, RX-type address, or register (2) - (12). *pgaddr*: RX-type address or register (2) - (12). *pcnum*: A constant, an expression, or register (2) - (12).
,ECB=*ecbaddr*	*ecbaddr*: RX-type address or register (2) - (12).
,PARAM=*paddr*	*paddr*: RX-type address or register (2) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

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

MF=(E,*listaddr*)
E specifies the execute form of the RESMGR macro and *listaddr* specifies the address of the parameter list.

RESUME — Resume Execution of a Suspended RB

Description

Resume an SRB

To resume or purge a suspended SRB, use the variation of the RESUME macro described under “RESUME — Resume or Purge a Suspended SRB” on page 231.

To resume a request block (RB) that was suspended through the SUSPEND macro, use this variation of the RESUME macro.

Environment

Requirements for the calling program's environment are:

Minimum authorization:	Supervisor state with PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Can be either enabled or disabled
Locks:	For RETURN=N, cannot hold local lock
Control parameters:	Must be in the caller's primary address space

Programming Requirements

The task to be resumed must be in the primary address space.

The caller must include the IHAPSA and CVT mapping macros.

Restrictions

The list and execute forms of the RESUME macro are not valid for resuming execution of a suspended RB.

Input Register Information

Before issuing the RESUME 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

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

When 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
2-3	Unchanged
4-5	Used as work registers by the system
6-10	Unchanged
11-14	Used as work registers by the system

© Copyright IBM Corp. 1988, 2000

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 RESUME macro is coded as follows:

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

TCB=(4) TCB=*tcbaddr*	**Default**: Register 4 contains TCB address. *tcbaddr*: A-type address or registers (2) - (12).
,RB=(5) ,RB=*rbaddr*	**Default**: Register 5 contains RB address. *rbaddr*: A-type address or registers (2) - (12).
,RETURN=Y ,RETURN=N	**Default**: RETURN=Y
,MODE=UNCOND ,MODE=COND	**Default**: MODE=UNCOND
,ASYNC=Y ,ASYNC=N	**Default**: ASYNC=N
,ASCB=*ascbaddr*	**Default**: ASCB address of the home address space. *ascbaddr*: RX-type address or registers (1) or (2) - (3) or (6) - (12).

Parameters

The parameters are explained as follows:

TCB=(4)
TCB=*tcbaddr*
Specifies the TCB address of the task to be resumed. Register 4 is the default; it is assumed to contain the TCB address.

Note: The TCB resides in storage below 16 megabytes.

,RB=(5)
,RB=*rbaddr*
Specifies the address of the RB to be resumed. Register 5 is the default; it is assumed to contain the address of the RB to be resumed.

Note: The RB resides in storage below 16 megabytes.

,RETURN=Y
,RETURN=N
Specifies whether control is to be returned to the caller (RETURN=Y) or not (RETURN=N). RETURN=N causes RESUME to make the specified TCB/RB dispatchable and gives the specified TCB/RB control directly. Only programs running under an SRB in primary ASC mode can issue RETURN=N. If you specify RETURN=N, you must also specify MODE=UNCOND and ASYNC=N and must not specify ASCB.

,MODE=UNCOND
,MODE=COND
If MODE=COND is specified, the action RESUME takes if the function cannot be completed synchronously depends on the ASYNC option. If ASYNC=Y is specified, RESUME makes a conditional attempt to acquire an SRB. If an SRB is available, it is scheduled to complete the RESUME function asynchronously. If ASYNC=N is specified explicitly or as a default and the RESUME cannot immediately complete the function, the system places return code 04 in register 15 and returns to the caller.

If MODE=UNCOND is specified, the action RESUME takes also depends on the ASYNC option. If ASYNC=Y is specified, RESUME makes an unconditional request for an SRB, and completes the RESUME function asynchronously. If ASYNC=N is specified explicitly or as a default, RESUME unconditionally obtains the CML lock of the ASCB whose TCB or RB is to be resumed. The TCB or RB is resumed before control returns to the caller.

,ASYNC=Y
,ASYNC=N
Specifies whether the RESUME is to be completed asynchronously (Y) or not (N).

,ASCB=*ascbaddr*
Specifies the address of the ASCB whose TCB or RB is to be resumed. The caller must establish current addressability to the address space before calling RESUME. If this option is not specified, the home address space is assumed. This option must be specified if ASYNC=Y is specified.

Note: The ASCB resides in storage below 16 megabytes.

ABEND Codes

070

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return Codes

When the RESUME macro returns control to your program, GPR 15 contains a hexadecimal return code.

Figure 47. Return Codes for the RESUME Macro for RBs

Return Code	Meaning and Action
00	**Meaning**: A normal, synchronous RESUME completed the function. **Action**: None.
04	**Meaning**: Environmental error. For MODE=COND and ASYNC=N, the RESUME cannot complete the function. For MODE=COND or MODE=UNCOND and ASYNC=Y, an SRB is completing the function asynchronously. **Action**: None required. However, you might take some action based upon your application.
08	**Meaning**: Environmental error. For MODE=COND and ASYNC=Y the SRB cannot be acquired and RESUME cannot complete the function. **Action**: None required. However, you might take some action based upon your application.

Example

Resume execution of the task specified in the address labeled CURRTCB. Use the request block address in register 5. Pass control back to the task (the issuer is currently in SRB mode and this step terminates SRB mode processing).

```
RESUME  TCB=CURRTCB,RB=(5),RETURN=N
```

RESUME — Resume or Purge a Suspended SRB

Description

Resume an RB

To resume an RB, use the variation of the RESUME macro described under “RESUME — Resume Execution of a Suspended RB” on page 227.

To resume or purge a suspended supervisor request block (SRB), use this variation of the RESUME macro. Optionally, the RESUME macro enables the caller to provide a fullword of data (the resume code) to the suspended SRB routine.

Environment

Requirements for the calling program are:

Minimum authorization:	Supervisor state or PSW key 0 - 7
Dispatchable unit mode:	SRB or task
Cross memory mode:	Any
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Can be enabled or disabled for interrupts
Locks:	Can hold no locks or can hold the local lock, the CML lock, the CMS lock, or the CPU lock
Control parameters:	Must be in the caller's primary address space or addressable through the caller's dispatchable unit access list (DU-AL)

Programming Requirements

Programming requirements for the calling program are:

- Before issuing the RESUME macro, ensure that the global symbol &SYSASCE is correctly set to indicate the ASC mode of your program. To test or set this global symbol, use the SYSSTATE macro.
- Programs in AR ASC mode must ensure that parameter addresses are ALET-qualified.

Restrictions

None.

Input Register Information

Before issuing the RESUME 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

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

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

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

© Copyright IBM Corp. 1988, 2000

2-13	Unchanged
14	Used as a work register by the macro
15	Return code

When control returns to the caller, the access registers contain:

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

Syntax

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

name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede RESUME.
RESUME	
ƀ	One or more blanks must follow RESUME.
SPTOKEN=*sptoken addr*	*sptoken addr*: RX-type address.
,PURGE=NO ,PURGE=YES	**Default**: PURGE=NO.
,RSCODE=*rscode addr*	*rscode addr*: RX-type address.
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

SPTOKEN=*sptoken addr*
Specifies the address of an 8-byte location that contains the system-provided suspend token. The suspend token identifies the SRB that is to be resumed or purged.

,PURGE=NO
,PURGE=YES
Indicates whether the system is to resume (PURGE=NO) or purge (PURGE=YES) the SRB. The default is PURGE=NO. A purged SRB never regains control and cannot be resumed. Do not use RSCODE with PURGE=YES.

,RSCODE=*rscode addr*
Specifies the address of a fullword where you can place a value that the system will return to the resumed SRB routine. Code RSCODE only if you also code PURGE=NO or take the default. If you omit RSCODE, the system returns a resume code of zero to the resumed SRB routine.

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

ABEND Codes

17

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return Codes

When the RESUME macro returns control to your program, GPR 15 contains a hexadecimal return code.

Figure 48. Return Codes for the RESUME Macro for SRBs

Return Code	Meaning and Action
00	**Meaning**: The system has scheduled the suspended SRB to be resumed. **Action**: None.
04	**Meaning**: The address space in which the suspended SRB would have executed has been scheduled for termination. The system will purge the suspended SRB. **Action**: None required. However, you might take some action based upon your application.
08	**Meaning**: The suspend token (SPTOKEN) does not identify a currently suspended SRB routine. The SRB may have already been resumed or purged. **Action**: None required. However, you might take some action based upon your application.
24	**Meaning**: System error. An error occurred while trying to resume the suspended SRB. The SRB cannot be resumed. **Action**: Retry the request.

Example

Resume the execution of a suspended SRB.

```
⋮
          RESUME   SPTOKEN=TOKEN,PURGE=NO,RSCODE=RCODE
⋮
          DS    F
RCODE     DC    X'99999999'
TOKEN     DS    CL8
⋮
```

RESUME—Resume or Purge an SRB (List Form)

For programs that require reentrant code, use the list form of the RESUME macro together with the execute form of the macro. The list form of the macro defines an area of storage that the execute form of the macro uses to store parameter values.

Syntax

The list form of the RESUME macro is valid only for resuming an SRB. It is written as follows:

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

MF=L	
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

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

MF=L
Requests the list form of RESUME.

RESUME—Resume or Purge an SRB (Execute Form)

For programs that require reentrant code, use the execute form of the RESUME macro together with the list form. The execute form of the macro stores the parameters into the storage area defined by the list form.

Syntax

The execute form of the RESUME macro is valid only for resuming an SRB. It is written as follows:

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

SPTOKEN=*sptoken addr* ,MF=(E,*cntl addr*)	*sptoken addr*: RX-type address. *cntl addr*: RX-type address or register (2) - (12)
,RSCODE=*rscode addr*	*rscode addr*: RX-type address.
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

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

,MF=(E,*cntl addr*)
Requests the execute form of RESUME. *cntl addr* must be the address of the parameter list provided by the list form of the macro.

RISGNL — Issue Remote Immediate Signal

Description

The RISGNL macro uses the emergency signal (EMS) order code of the signal processor (SIGP) instruction to invoke the execution of a specified software program on a specific processor in a multiprocessing configuration. The program may be requested to execute in parallel or serially with the function requesting the program. The specified software program (receiving routine) gets control disabled, in key 0, and supervisor state. The receiving routine cannot enable for I/O or external interrupts, request locks, or issue SVCs. In addition, the receiving routine must return via the address in register 14.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or secondary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No locks required
Control parameters:	None

Programming Requirements

- The receiving routine must be loaded into page-fixed, common storage.
- The caller must include the CVT mapping macro.

Restrictions

If the receiving routine establishes an FRR, the FRR should not depend on the storage areas passed to it by the signalling routine. When alternate CPU recovery (ACR) is active, the signalling routine might receive control from RISGNL before the receiving routine's FRR gets control. IBM recommends that the receiving routine's FRR access only storage areas owned independently of the signalling routine. If the receiving routine's FRR attempts to access storage and the signalling routine has already freed the storage, the FRR might abnormally end.

Input Register Information

Before issuing the RISGNL 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

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

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-10	Unchanged

© Copyright IBM Corp. 1988, 2000

11-12	Used as work registers by the system
13	Unchanged
14	Address of calling program
15	Return code

Performance Implications

None.

Syntax

The RISGNL macro is written as follows:

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

PARALLEL SERIAL	
,CPU=*PCCA addr*	*PCCA addr*: RX-type address, or register (1).
,EP=*entry name addr*	*entry name addr*: RX-type address, or register (12).
,PARM=*parm addr*	*parm addr*: RX-type address, or register (11).

Parameters

The parameters are explained as follows:

PARALLEL
SERIAL
Specifies that control is to be returned to the caller when the specified receiving routine has been given control (PARALLEL) or has completed execution (SERIAL) on the designated processor.

,CPU=*PCCA addr*
Specifies the address of the physical configuration communication area (PCCA) of the processor on which the function is to be performed.

Note: The PCCA must reside in 24-bit addressable storage.

,EP=*entry name addr*
Specifies the address of the receiving routine to be executed on the specified processor. The receiving routine will get control in the same addressing mode as the macro issuer.

,PARM=*parm addr*
Specifies the address of a user-defined fullword parameter to be passed to the receiving routine. When the receiving routine receives control, general purpose register one points to a fullword parameter.

ABEND Codes

07B

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return Codes

When the RISGNL macro returns control to your program, GPR 15 contains a hexadecimal return code.

Figure 49. Return Codes for the RISGNL Macro

Return Code	Meaning and Action
00	**Meaning**: Specified receiving routine has been given control or has completed execution, as requested. **Action**: None.
04	**Meaning**: Environmental error. Function not initiated because addressed processor not online. If it appeared to be online, it is no longer in the configuration. **Action**: None required. However, you might take some action based upon your application.
14	**Meaning**: Environmental error. Function not initiated because addressed processor was taken offline during RISGNL processing. **Action**: None required. However, you might take some action based upon your application.

Example 1

The routine whose address is in register 12 is to be given control on the processor whose PCCA address is in register 1. Return control to the caller when the specified receiving routine has been given control.

```
RISGNL PARALLEL,CPU=(1),EP=(12)
```

Example 2

The routine whose address is in register 12 is to be given control on the processor whose PCCA address is in register 1. The routine will complete before the caller of RISGNL receives control again. Register 11 contains the address of a parameter to be passed to the receiving routine.

```
RISGNL SERIAL,CPU=(1),EP=(12),PARM=(11)
```

SCHEDIRB — Schedule IRB

Description

Use the SCHEDIRB macro to initialize and schedule asynchronous exits.

Using the SCHEDIRB macro, you can:

- Schedule the asynchronous exit to run under any task in the current address space.
- Schedule the asynchronous exit to run prior to any RB under the current task in the current address space.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB. Task mode is required if the RBPTR keyword is specified.
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	Local lock must be held for the address space where the asynchronous exit will get control.
Control parameters:	Must be in the primary address space

Programming Requirements

None.

Restrictions

None.

Input Register Information

Before issuing the SCHEDIRB 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
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

© Copyright IBM Corp. 1988, 2000

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 SCHEDIRB macro is written as follows:

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

EPPTR=*ep addr*	*ep addr*: RX-type address or address in register (2) - (12).
,TCBPTR=*tcb addr* ,RBPTR=*rb addr* ,IQEPTR=*iqe addr*	*tcb addr*: RX-type address or address in register (2) - (12). *rb addr*: RX-type address or address in register (2) - (12). *iqe addr*: RX-type address or address in register (2) - (12). RETCODE is the only parameter you can specify with IQEPTR.
,MODE=PROB ,MODE=SUPR	**Default**: MODE=PROB
,KEY=PROP ,KEY=SUPR	**Default**: KEY=PROP
,SVAREA=NO ,SVAREA=YES	**Default**: SVAREA=NO SVAREA=YES can only be specified with TCBPTR
,PARAMPTR=*parm addr*	*parm addr*: RX-type address or address in register (2) - (12).
,RETCODE=*ret code*	*ret code*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

EPPTR=*ep addr*
Specifies a required input parameter containing the 31-bit entry point address of the asynchronous exit routine. The exit routine will get control in 31-bit addressing mode.

,TCBPTR=*tcb addr*
,RBPTR=*rb addr*
,IQEPTR=*iqe addr*
TCBPTR specifies an input parameter containing the address of a TCB in the current address space. The asynchronous exit will run under the TCB specified. Use TCBPTR when you want your asynchronous exit to run under a particular task.

RBPTR specifies an input parameter containing the address of an RB under the current TCB in the current address space. The asynchronous exit will run directly prior to the specified RB. Use RBPTR when you want your asynchronous exit to run before a particular RB.

IQEPTR specifies an input parameter containing the address of an IQE initialized using the CIRB macro. If you specify IQEPTR, SCHEDIRB will not obtain a new IQE/IRB pair for scheduling the asynchronous exit.

This option is only valid if you use the CIRB macro to create and initialize an IRB for the asynchronous exit. For more information on using the CIRB macro, see the *OS/390 MVS Programming: Authorized Assembler Services Guide*.

RETCODE is the only optional parameter you can specify with IQEPTR.

Notes:

1. The caller must be in task mode to use the RBPTR parameter.
2. You cannot specify the current RB (the RB of the calling program), or an RB that is not on the current task's RB chain on the RBPTR parameter.
3. For best results with the RBPTR parameter, make sure that the calling program is running under an IRB. If the calling program is not running under an IRB, and the system has suppressed asynchronous exits or is in process-must-complete mode, the calling program will get a nonzero return code.

   You can make sure that the calling program is running under an IRB by first invoking the SCHEDIRB macro with the TCBPTR option, or by invoking the STIMER macro.

,MODE=PROB
,MODE=SUPR
Specifies whether the asynchronous exit routine is to operate in problem program (PROB) or supervisor (SUPR) state.

,KEY=PROP
,KEY=SUPR
Specifies that the asynchronous exit routine run in the key propagated from the target TCB (PROP) or in supervisor key zero (SUPR).

,SVAREA=NO
,SVAREA=YES
Specifies whether to obtain a 72-byte register save area from the virtual storage assigned to the task specified by the TCBPTR keyword. SVAREA=YES can only be specified with TCBPTR.

,PARAMPTR=*parm addr*
Specifies an input parameter containing the address of the parameter list to be passed to the asynchronous exit routine.

RETCODE=*ret code*
Specifies a storage location or register where the system is to store the return code. The return code is also in GPR 15.

ABEND Codes

The SCHEDIRB macro issues the X'AC7' abend code. See *OS/390 MVS System Codes* for more information.

Return and Reason Codes

When the SCHEDIRB macro returns control to your program, GPR 15 and *retcode*, if you specified RETCODE, contains a return code. The return codes are shown in the following table.

Figure 50. Return Codes for the SCHEDIRB Macro

Return Code Hexadecimal (Decimal)	Meaning and Action
00	**Meaning**: Successful completion. **Action**: None.
08 (8)	**Meaning**: Program error. The caller invoked SCHEDIRB with the RBPTR parameter, but RBPTR is not valid when the system has suppressed asynchronous exits unless the calling program is running under an IRB. **Action**: Ensure that the calling program is running under an IRB when you invoke SCHEDIRB by first invoking the SCHEDIRB macro with the TCBPTR option or by invoking the STIMER macro.
0C (12)	**Meaning**: Program error. The caller invoked SCHEDIRB with the RBPTR parameter, but RBPTR is not valid when the current task is in process-must-complete mode unless the calling program is running under an IRB. **Action**: Ensure that the calling program is running under an IRB when you invoke SCHEDIRB by first invoking the SCHEDIRB macro with the TCBPTR option or by invoking the STIMER macro.
10 (16)	**Meaning**: Program error. **See Action**. **Action**: Make sure that your program does not specify one of the following on the RBPTR parameter: • The current RB (the RB of the caller). • An RB that is not on the current task's RB chain.

SCHEDIRB—List Form

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

name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede SCHEDIRB macro.
SCHEDIRB macro	
␣	One or more blanks must follow SCHEDIRB macro.

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 SCHEDIRB 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 SCHEDIRB 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.

SCHEDIRB—Execute Form

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

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

EPPTR=*ep addr*	*ep addr*: RX-type address or address in register (2) - (12).
,TCBPTR=*tcb addr* ,RBPTR=*rb addr* ,IQEPTR=*iqe addr*	*tcb addr*: RX-type address or address in register (2) - (12). *rb addr*: RX-type address or address in register (2) - (12). *iqe addr*: RX-type address or address in register (2) - (12). RETCODE is the only parameter you can specify with IQEPTR.
,MODE=PROB ,MODE=SUPR	**Default**: MODE=PROB
,KEY=PROP ,KEY=SUPR	**Default**: KEY=PROP
,SVAREA=NO ,SVAREA=YES	**Default**: SVAREA=NO SVAREA=YES can only be specified with TCBPTR
,PARAMPTR=*parm addr*	*parm addr*: RX-type address or address in register (2) - (12).
,RETCODE=*ret code*	*ret code*: RX-type address or address in 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 SCHEDIRB macro with the following exception:

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

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

COMPLETE, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.

SCHEDULE — Schedule a Service Request Block (SRB)

IBM recommends that you use the IEAMSCHD macro rather than SCHEDULE.

Description

Use the SCHEDULE macro to schedule a service request block (SRB) for asynchronous execution. The SRB may be scheduled for execution in any address space and may be scheduled at either global or local priorities.

A global SRB has a priority that is greater than, and independent of, any address space priority. A local SRB has a priority within the specific address space in which it executes, but still has a priority greater than that of any task within the address space.

On the SRB parameter, you specify the address of the SRB. See the section on scheduling SRBs in *OS/390 MVS Programming: Authorized Assembler Services Guide* for information about obtaining storage for the SRB and initializing the fields in the SRB.

For information on using this macro on an MVS/SP version other than the current version, see “Selecting the Macro Level” on page 7.

Environment

The requirements for the caller when MODE=FULLXM is specified on the SCHEDULE macro are:

Minimum authorization:	Supervisor state and PSW key zero
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any
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
Storage requirements:	The SRB to be scheduled must be in fixed, commonly addressable storage, with any storage key (0-7).

The requirements for the caller when MODE=NONXM is specified on the SCHEDULE macro are:

Minimum authorization:	Supervisor state and PSW key zero
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	Any locks may be held
Storage Requirements:	The SRB to be scheduled must be in fixed, key 0, common storage.

Programming Requirements

The scheduling program must obtain 44 bytes of storage for the SRB and initialize certain fields, as described in the section on scheduling SRBs in *OS/390 MVS Programming: Authorized Assembler Services Guide*.

The scheduling program that builds the SRB must include the IHASRB mapping macro, the CVT mapping macro with DSECT=YES specified, and the IHAPSA mapping macro.

© Copyright IBM Corp. 1988, 2000

Restrictions

Address space resource managers cannot use the STOKEN parameter.

Input Register Information

Before issuing the SCHEDULE 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

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 SCHEDULE macro is written as follows:

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

SRB=*SRB addr*	*SRB addr*: RX-type address, or register (1) or (2) - (12).
,SCOPE=LOCAL ,SCOPE=GLOBAL	**Default**: SCOPE=LOCAL
,LLOCK=NO ,LLOCK=YES	**Default**: LLOCK=NO
,MODE=NONXM ,MODE=FULLXM	**Default**: MODE=NONXM Do not specify DISABLED or STOKEN when specifying MODE=FULLXM.
,FRR=NO ,FRR=YES	**Default**: FRR=NO

```
,DISABLED

,STOKEN=stoken addr          stoken addr: RX-type address

,FEATURE=CRYPTO
```

Parameters

The parameters are explained as follows:

SRB=*SRB addr*
Specifies the address of the service request block (SRB).

,SCOPE=LOCAL
,SCOPE=GLOBAL
Specifies whether the service is to be scheduled at a local or global priority.

,LLOCK=NO
,LLOCK=YES
Specifies whether the SRB is to receive control with the LOCAL lock held.

Note: CML (cross memory local) lock means the local lock of an address space other than the home address space. LOCAL lock means the local lock of the home address space. When written in lower case, local lock means any local-level lock, either the LOCAL or a CML lock.

,MODE=NONXM
,MODE=FULLXM
Specifies whether or not the SRB routine receives a copy of the scheduling program's dispatchable unit access list (DU-AL), and it receives control in the scheduling program's current cross memory environment.

When you specify NONXM, the SRB routine receives control in noncross memory mode, and it receives an empty DU-AL. The SRB's primary, secondary, and home address spaces are all equal to the contents of SRBASCB.

When you specify FULLXM:

- The SRB routine will be able to access a copy of the scheduling program's DU-AL, with the exception of any subspace entries in the scheduling program's DU-AL. The system does not copy subspace entries. If the scheduling program establishes addressability to any new data spaces after the SRB is scheduled, the SRB routine will not have access to the new data space. See the discussion on access lists in *OS/390 MVS Programming: Extended Addressability Guide* for more details about how the system copies a DU-AL.
- The SRB routine will receive control in the scheduling program's current cross memory environment.
- Addressing is the following:
  - Primary is the scheduling program's primary
  - Secondary is the scheduling program's secondary
  - Home is the scheduling program's home

When you specify FULLXM, a DU-AL with more than 256 entries is not available to the scheduled SRB routine until the SRB routine is dispatched. If an error occurs before the SRB routine is dispatched, the DU-AL might not be available to the SRB routine's FRR.

When you specify FULLXM, you cannot:

- Specify STOKEN or DISABLED with this parameter.
- Use the contents of SRBASCB because it does not contain relevant information for this type of SCHEDULE invocation.

,FRR=NO
,FRR=YES
Specifies whether the SRB is to receive control with recovery established. If FRR=YES is specified, the user must place the address of the FRR in the field SRBFRRA of the SRB. Before the SRB receives control, the system adds the FRR to the FRR stack. When you specify YES, the system passes a 24-byte FRR parameter area address to the SRB routine in register 2. If the scheduling program specifies MODE=FULLXM and FRR=YES, the recovery routine will be established with SETFRR MODE=FULLXM. If the scheduling program specifies MODE=NONXM and FRR=YES, the recovery routine will be established with SETFRR MODE=HOME.

,DISABLED
Specifies that the calling program is running disabled. DISABLED should be specified only when the calling program is disabled for I/O or external interrupts.

,STOKEN=*stoken addr*
Specifies the address of the 8-byte STOKEN of the address space in which the SRB routine is to run. SCHEDULE verifies that SRBASCB represents the same address space that the STOKEN identifies and that the address space is still active. If the address space is different or the address space is not active, the system abends the caller. This action prevents a scheduled SRB from running in an address space other than the one intended.

,FEATURE=CRYPTO
Specifies that the SRB routine must run on a processor that has an Integrated Cryptographic Feature (ICRF) associated with it. When you specify this parameter, the system assigns the correct processor affinity for the routine and overrides any affinity assigned for the routine in the SRBCPAFF field of the SRB. Use this parameter only for routines whose exclusive purpose is to encrypt or decrypt data.

ABEND Codes

If STOKEN is specified, the caller might encounter a X'075' abend. If STOKEN is not specified, the caller might encounter a X'08C' abend. See *OS/390 MVS System Codes* for an explanation and programmer responses for these codes.

Return and Reason Codes

There are no return codes from the SCHEDULE macro. If the SCHEDULE fails, an abnormal termination may occur.

Example 1

Schedule an SRB at a global priority.

```
SCHEDULE SRB=(1),SCOPE=GLOBAL
```

Example 2

Schedule an SRB at a local priority.

```
SCHEDULE SRB=(1),SCOPE=LOCAL
```

Example 3

Schedule an SRB at a global priority specifying that the SRB is to receive control with the LOCAL lock held and recovery established. The issuer of the SCHEDULE macro is disabled.

```
SCHEDULE SRB=(1),SCOPE=GLOBAL,LLOCK=YES,FRR=YES,DISABLED
```

Example 4

Schedule an SRB at a local priority specifying that the SRB is to receive control with the LOCAL lock held and recovery established. The SRBASCB field of the SRB points to the home address space ASCB. The STOKEN parameter also identifies the home address space.

```
         MVC     SRBASCB,PSAAOLD
         ALESERV EXTRACTH,STOKEN=MYSTOKEN
         SCHEDULE SRB=(1),LLOCK=YES,FRR=YES,STOKEN=MYSTOKEN
           .
           .
MYSTOKEN DS 2F
```

Example 5

Schedule an SRB at a local priority specifying that the SRB is to receive control with the scheduling program's cross memory environment, a copy of the caller's DU-AL, and recovery established. The SRB will have addressability to the data space represented by DSSTOKEN.

```
         ALESERV ADD,STOKEN=DSSTOKEN,ALET=DSALET
         SCHEDULE SRB=(1),MODE=FULLXM,FRR=YES
           .
           .
DSSTOKEN DS 2F
DSALET   DS 1F
```

Example 6

Schedule an SRB at a local priority specifying that the SRB routine is to run on a processor with an ICRF associated with it.

```
SCHEDULE SRB=(1),SCOPE=LOCAL,FEATURE=CRYPTO
```

SCHEDXIT — Schedule an Exit Routine for Execution

Description

Note: IBM recommends that you use the SCHEDIRB macro rather than SCHEDXIT to request asynchronous exits.

The SCHEDXIT macro schedules an asynchronous exit routine for execution under a specific task.

Before using this macro, read the description of using asynchronous exits in the *OS/390 MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Minimum authorization:	Supervisor state with PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Local lock held
Control parameters:	None

Programming Requirements

If the IQE resides below 16 megabytes, the IQE address that is passed must be a 31-bit address with the high-order byte of the address set to zero.

The caller must have addressability to the address space on which the exit routine is to be dispatched.

Restrictions

None.

Input Register Information

Before issuing the SCHEDXIT 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

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

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

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

© Copyright IBM Corp. 1988, 2000

Performance Implications

None.

Syntax

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

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

IQE=*iqe-address*	*iqe-address*: RX-type address or register (2) - (12).

Parameters

The parameter is explained as follows:

IQE=*iqe-address*
Specifies the address of the interrupt queue element (IQE) that defines the task under which the exit routine will execute.

ABEND Codes

00A

See *OS/390 MVS System Codes* for an explanation and programmer responses for this code.

Return and Reason Codes

None.

SDUMP — Dump Virtual Storage

Description

Note: IBM recommends that you use the SDUMPX macro rather than SDUMP.

The SDUMP macro invokes SVC dump to provide a fast unformatted dump of virtual storage to a data set. It is intended for use by authorized routines that encounter errors. If your program is in primary ASC mode, you can use either SDUMP or SDUMPX. If your program runs in access register (AR) mode, use SDUMPX instead of SDUMP. SDUMPX provides all of the functions of SDUMP, as well as some that SDUMP does not offer, but generates code and addresses that are appropriate for AR mode.

You cannot use the SDUMP macro to dump data space storage. To dump data space storage, issue SDUMPX.

There are two phases in SVC dump processing:

- The capture phase, in which all the data for the dump is captured
- The writing phase, in which the data is written to the dump data set.

The caller can initiate an SVC dump in an address space other than the primary. A branch entry is available for callers who wish a dump of their own or another address space, but cannot issue an SVC.

When you request a dump of virtual storage, the combination of parameters you code determines whether MVS produces either a **scheduled** (asynchronous) or a **synchronous** SVC dump. You might make different design decisions for your program based on the type of dump that MVS produces. Read the information about dumping virtual storage in *OS/390 MVS Programming: Authorized Assembler Services Guide* for the parameter combinations that produce each type of dump, and for guidance about designing your program to handle each type.

For information about how to select this macro for an MVS/SP version other than the current version, see “Selecting the Macro Level” on page 7.

Except for the data control block (DCB) parameter, all input parameters to this macro can reside in storage above 16 megabytes if the caller is executing in 31-bit addressing mode.

You can produce reentrant code using the standard form of SDUMP if you do not specify parameters other than the following:

SDATA
TYPE
HDR
ID
BRANCH
SUSPEND
QUIESCE
BUFFER
PLISTVER

Programs in page-protected storage (such as the nucleus, PLPA, and MLPA) can issue the standard form of the SDUMP macro without causing a protection exception. However, IBM recommends using the list and execute forms of SDUMP for programs in page-protected storage.

© Copyright IBM Corp. 1988, 2000

Environment

The requirements for the caller with **BRANCH=NO** are:

Minimum authorization:	Any one or more of the following: Supervisor state PSW keys 0 - 7 APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters and all areas the parameter list points to (except the DCB, ECB, and SRB) must be addressable from the current address space. The DCB must be addressable in the home address space. The ECB and SRB must be addressable from each address space included in the dump. The SRB must be addressable from the address space in which it will run.

The requirements for the caller with **BRANCH=YES** are:

- Be in SRB mode
- Hold any lock
- Have an enabled-unlocked-task FRR on the FRR stack

Assuming that one of the above conditions has been satisfied, other requirements for callers with BRANCH=YES are:

Minimum authorization:	All of the following: Supervisor state PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Control parameters and all areas the parameter list points to (except the DCB, ECB, and SRB) must be addressable from the current address space. The DCB must be addressable in the home address space. The ECB and SRB must be addressable from each address space included in the dump. The SRB must be addressable from the address space in which it will run.

Programming Requirements

To generate reentrant code, code the list and execute forms of the SDUMP macro. Because the execute form of the macro is dependent on the length determined by the list form, the list form must appear before the execute form in a reentrant program.

Callers can determine the length of the parameter list by using the following programming technique to calculate the amount of storage needed for only those options specified for the SDUMP macro:

```
SDUMPBEG SDUMP SDATA=(SUM),SUMLIST=SLIST,MF=L
SDUMPEND EQU   *
SDUMPLEN DC    A(SDUMPEND-SDUMPBEG)
```

Callers that issue SDUMP with BRANCH=YES must include the CVT mapping macro.

Restrictions

For SVC entry, the caller cannot have an EUT FRR established.

Input Register Information

Upon invocation, general purpose register (GPR) 13 must contain the address of a 72-byte savearea if BRANCH=YES is specified.

Output Register Information

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

When 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 in bits 24-31. If the return code is X'08', and you specified the FAILRC parameter, GPR 15 also contains a reason code in bits 16-23.

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 SDUMP macro is written as follows:

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: From 1 to 100 characters. *dump title addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,ASID=*ASID addr* ,ASIDLST=*list addr*	*ASID addr*: A-type address, or register (2) - (12). *list addr*: RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code*: Any of the following, separated by commas: XMEM, XMEME, NOLOCAL, FAILRC **Note**: XMEM and XMEME are mutually exclusive codes.
,PLISTVER=1	*decimal digit 1*: Use up to a 68-byte parameter list.

,PLISTVER=2	*decimal digit 2*: Use 128-byte parameter list. **Default**: PLISTVER=1, unless you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFS, SDATA=NODEFS, or SDATA=IO, in which case the default is PLISTVER=2.
,SYMREC=*symrec addr*	*symrec addr*: RX-type address, or register (2) - (12).
,ID='*identifier*' ,IDAD=*identifier addr*.	*identifier*: From 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,ECB=(*ecb addr*) ,SRB=(*srb addr*)	*ecb addr*: A-type address, or register (2) - (12). *srb addr*: A-type address, or register (2) - (12). **Notes:** 1. SVC dump posts the ECB at the completion of the capture phase unless the DCB parameter is specified with the ECB parameter. If you specify both the DCB and ECB parameters, the ECB is posted at the completion of the writing phase. 2. SVC dump schedules the SRB at the completion of the capture phase unless the DCB parameter is specified with the SRB parameter. If you specify both the DCB and SRB parameters, the SRB is scheduled at the completion of the writing phase.
,SDATA=*(sdata options)*	*sdata options*: Any combination of the following, separated by commas: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing SDUMP causes ALLPSA, SQA, IO, and SUMDUMP storage areas to be dumped unless excluded by NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP. 2. The PSA and IO options are not required unless NODEFAULTS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS has been specified.
,STORAGE=*(strt addr,end addr)* ,LIST=*list addr* ,LISTA=*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**: Specify one or more pairs of addresses, separated by commas.
,SUBPLST=*subpool id list addr*	*subpool id list addr*: RX-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr*: RX-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default**: BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default**: QUIESCE=YES
,BRANCH=NO ,BRANCH=YES	**Default**: BRANCH=NO **Note**: If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default**: SUSPEND=NO

,SUMLIST=*list addr* ,SUMLSTA=*list addr*	*list addr*: RX-type address, or register (2) - (12).

Parameters

The parameters are explained as follows:

HDR=*'dump title'*
HDRAD=*dump title addr*

Specifies the title or address of the title to be used for the dump. If HDR is specified, the title must be 1-100 characters enclosed in apostrophes, although the apostrophes do not appear in the actual title. If HDRAD is specified, the first byte at the indicated address specifies the length of the title in bytes.

If the length of the title is greater than 100, SVC dump issues an abend with a completion code of X'233', reason code X'14', then returns to the caller with a return code of 8. If the length of the title is zero, SVC dump continues processing as if the HDR or HDRAD parameter was not specified.

If these keywords are specified with BRANCH=YES or ASID/ASIDLST (that is, to cause a scheduled dump), the system moves the title to SVC dump storage before it returns control to the caller. There is no requirement to synchronize with the completion of the dump.

,DCB=*dcb addr*

Specifies the address of a previously opened data control block (DCB) for the data set that is to contain the dump. If this parameter is omitted, one of the SYS1.DUMP data sets is used. When you specify the DCB parameter, the dump contains data from only the requestor's home address space. The DCB must be addressable from the home address space. The control blocks built by OPEN must also be addressable from the home address space. The DCB must support EXCP. You must specify the following parameters on the DCB macro: RECFM=FB, LRECL=4160, and BLKSIZE=4160.

The DCB must reference device types supported by SVC dump. Eligible device types are unlabeled 9-track 2400-series tape devices (or tape devices compatible with the 2400-series) and any direct access devices supported by the system that have a track size of at least 4160 bytes. (4160 bytes equals 1 SVC dump output record.) SVC dump does not support secondary extents on DCB data sets.

SVC dump does not close the dump data set. Use the CLOSE macro to close the data set and cause an end-of-file mark or a tape mark to be placed after the dump data. SVC dump sets up the DCB so that CLOSE works correctly and positions the end-of-file mark or tape mark at the correct place on the data set. For tape data sets, you can write a tape mark to separate multiple dumps without using the CLOSE macro.

Because it is the caller's responsibility to close the dump data set and the data set may be closed only after all the data has been written to it, the caller needs to receive notification when the dump writing phase is complete. Therefore, if you specify the DCB parameter with the ECB parameter, the system posts the ECB at the completion of the dump writing phase. The ECB parameter is required when a DCB is provided for scheduled dumps. If an ECB is not provided with the DCB for a synchronous dump, SVC dump returns to the caller at the completion of the dump writing phase. See *OS/390 MVS Programming: Authorized Assembler Services Guide* for descriptions of scheduled and synchronous dumps.

,ASID=*ASID addr*
,ASIDLST=*list addr*

Specifies the address of a halfword or a list of halfwords containing the hexadecimal address space identifier of an address space to be dumped. If register notation is used, the low-order halfword of the register contains the address space identifier of the address space to be dumped. If both parameters are omitted, the primary address space is dumped. If 0 is specified for the address space identifier, a dump is scheduled for the home address space of the issuer of the SDUMP macro. No private area storage

is included in the dump for the specified address space if either of the following events occurred:

- No SDATA parameters were specified that apply to the private area of the requested address space.
- The CHNGDUMP operator command was used to set an overriding parameter in the system dump options list that limits SVC dumps to areas outside of the private area.

The ASID list can contain a maximum of 15 address space identifiers. The high-order bit of the halfword containing the last identifier of the list must be set to 1, and all other high-order bits must be set to 0.

,TYPE=XMEM
,TYPE=XMEME
,TYPE=NOLOCAL
,TYPE=FAILRC

Specifies that the caller's cross memory mode determines the address spaces to dump (XMEM or XMEME) or that the caller cannot allow SDUMP to obtain a local lock (NOLOCAL) or that SVC dump should return a reason code with the return code to the DUMP command processor when the requested dump was not taken (FAILRC).

XMEM requests SVC dump to use the caller's cross memory mode at the time the SDUMP macro is executed.

XMEME requests SVC dump to use the caller's cross memory mode at the time of the error for which the dump is being taken. The home address space is dumped for both keywords. The relevant primary and secondary address spaces are also dumped if they are unique. If a cross memory local lock was held, the address space whose local lock is held is also dumped.

NOLOCAL indicates that the caller is in an environment where SDUMP cannot hold a local lock. This option has meaning only when BRANCH=YES is specified and the caller is enabled and unlocked (for example, the caller has an enabled unlocked task FRR established or is in SRB or cross memory mode).

FAILRC requests that the caller receive special information from SVC dump whenever the dump fails. Some information is already placed in SDWASDRC as a result of the SVC dump failure. When the caller receives control again after a dump failure (return code 8) and the caller has specified TYPE=FAILRC, the reason code is combined with the return code and passed to the caller in either register 15 or the ECB, or through the IHASDST mapping macro if the SRBPARM area was provided for an SRB. The reason code is in bits 16-23; the return code is in bits 24-31. When the return code is in the ECB, the POST flag is set on. SDUMP passes back a return code in register 15 and places the reason code in the SDWA. The reason code explains why the dump failed.

,PLISTVER=1
,PLISTVER=2

Specifies the length of the parameter list used. When PLISTVER=1 is specified, SDUMP uses a parameter list of up to 68 bytes. PLISTVER=2 specifies a 128-byte parameter list. The default is PLISTVER=1, unless you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFAULTS, SDATA=NODEFAULTS, or SDATA=IO, in which case the default is PLISTVER=2.

,SYMREC=*symrec addr*

Specifies the address of a valid symptom record for DAE to use for dump suppression. DAE suppresses the SVC dump if the primary symptom string found in the symptom record matches previously known symptoms, and, suppression has been enabled by the installation.

The caller must build the symptom record and fill in at least the ‘SR’ identifier and the primary symptom string, which should uniquely identify the error.

SVC dump issues an abend with a completion code of X'233', reason code X'9C', then returns to the caller with a return code of 8 if the symptom record identifier is not

'SR', if the offset and length of the primary symptom string are not initialized, or if the first byte of the symptom record and the last byte of the secondary symptom string are not addressable.

SVC dump does not include the symptom record in the dump. The caller can use the SUMLIST keyword to include the symptom record in the dump.

See the dump analysis and elimination (DAE) section in *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information on symptom strings and how to build them.

The ADSR macro maps the symptom record. See *OS/390 MVS Data Areas, Vol 1 (ABEP-DALT)* for a macro mapping of the ADSR.

,ID=*'identifier'*
,IDAD=*identifier addr*

Specifies an identifier that is included in dump message IEA911E, which is issued at the completion of the dump. The identifier must be from one to 50 printable characters. If ID is specified, the identifier must be enclosed in apostrophes, although the apostrophes do not appear in the actual identifier. If IDAD is specified, the first byte at the indicated address specifies the length of the identifier in bytes. If the length of the identifier is greater than 50, SVC dump issues an abend with a completion code of X'233', reason code X'8C', then returns to the caller with a return code of 8. If the length of the identifier is zero, SVC dump continues processing as if the ID or IDAD parameter was not specified.

,PSWREGS=*list addr*

Specifies a PSW or register area to be passed to SVC dump. This area may contain a PSW, control registers 3 and 4, all the general purpose registers (GPRs), and all the access registers (ARs). When PSWREGS is specified, SVC dump includes the following information in the summary dump portion of the dump:

- The PSWREGS parameter list
- If the PSW is provided, 4K of storage before and 4K after the PSW address from the primary address space.
- 4K of storage before and 4K of storage after each of the GPRs from the primary and secondary address spaces.
- If the ARs are provided, they qualify the addresses of the area that includes the 4K of storage before and 4K of storage after each of the GPRs. GPRs will be used to locate storage; ARs (if provided along with a PSW in AR mode) will be used to identify the source address space or data space.

Note: If the control registers are provided, they will be used to determine the primary and secondary address spaces. If no control registers are provided, then the storage will be dumped from the caller's primary and secondary address spaces.

The PSWREGS parameter allows programs running in a nonabend environment, where there is no SDWA, to request SVC dump and dump suppression services similar to those available in an abend environment, where an SDWA is present.

The parameter list for the PSWREGS parameter must reside in the address space currently addressable by SVC dump. The caller must provide at least the length and the mask field. Each bit in the mask refers to a data area. If a bit is set, SVC dump expects that the data area is supplied. If a mask bit is off and any lower-order mask bit is on, the corresponding storage area must be included in the parameter list. If a mask bit is off, but no lower-order mask bits are set, then the storage area need not be supplied. The following diagram describes the parameter list:

Figure 51 (Page 1 of 2). PSWREGS Parameter List

Offset in Hex	Length	Field Description
00	2	The total length of the PSWREGS parameter list

Figure 51 (Page 2 of 2). PSWREGS Parameter List

Offset in Hex	Length	Field Description
02	2	Bit mask describing data areas included in the PSW/register area
	1...	Bit 1: On - The PSW is included in the PSW/register area
	.1..	Bit 2: On - Control registers 3 and 4 are included in the PSW/register area
	..1.	Bit 3: On - General purpose registers are included in the PSW/register area
	...1	Bit 4: On - ARs are included in the PSW/register area. Bits 5 - 16: Initialize these bits to zero.
04	8	PSW: Data only supplied if the PSW mask bit is set
0C	8	Control registers 3 and 4: Data only supplied if mask bit is set.
14	64	General purpose registers 0 - 15: Data only supplied if mask bit is set.
54	64	ARs 0 - 15: Data only supplied if mask bit is set.

,ECB=(*ecb addr***)**
,SRB=(*srb addr***)**

Specifies how the system should synchronize your program with dump processing.

ECB specifies that the system should post the event control block (ECB) indicated by *ecb addr*. The system normally posts the ECB at the completion of the capture phase. However, if you specify the DCB parameter with the ECB parameter, the system posts the ECB at the completion of the dump writing phase. If the capture phase is not successful, the system posts the ECB at the completion of SVC dump processing.

If an A-type address is specified, *ecb addr* specifies the address of a fullword containing the address of the ECB. If a register operand is used, the register must contain the actual address of the ECB. If this parameter is omitted, the caller is not notified of the completion of the capture phase. The fullword and the ECB must be addressable from the home address space. The fullword address that points to the ECB must be a 24-bit or 31-bit address.

SRB specifies that the system should schedule the service request block (SRB) indicated by *srb addr*. The system normally schedules the SRB at the completion of the capture phase. However, if you specify the DCB parameter with the SRB parameter, the system schedules the SRB at the completion of the dump writing phase. If the capture phase is not successful, the system schedules the SRB at the completion of SVC dump processing.

When the caller builds the SRB, the caller may pass the address of a status area in the SRBPARM field. This status area, SDSTATUS, is mapped by IHASDST. SVC dump passes information about the dump to the SRB routine by means of this status area. If SVC dump schedules the SRB at the completion of the capture phase, the name of the dump data set is not passed to the caller.

Note: If you want SVC dump to pass the name of the dump data set to the caller, use the SDUMPX macro and specify SRB=(*srb addr*,WRITE).

,SDATA=*(sdata options)*

Specifies the system is to dump the following:

Option	Data to be Dumped
ALLNUC	The DAT-ON and DAT-OFF nuclei. The read-only (page-protected) area of the nucleus and the DAT-OFF nucleus is not included in the dump unless this keyword is specified.
ALLPSA	All of the prefixed storage areas (PSAs) in the system.
CSA	The CSA and ECSA subpools (subpools 227, 228, 231 and 241).
GRSQ	Global resource serialization control blocks.

LPA
: The active link pack area modules and SVCs for each address space being dumped.

LSQA
: The LSQA and ELSQA for each address space being dumped (subpools 203-205, 213-215, 223-225, 233-235, and 253-255).

NOALLPSA or NOALL
: The PSA for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

NOSQA
: The system queue area is not dumped.

NOSUMDUMP or NOSUM
: A summary dump is not included in the SVC dump.

NUC
: The non-page-protected areas of the DAT-ON nucleus. (The ALLNUC parameter must be specified to obtain the entire nucleus, including the page-protected areas of the DAT-ON nucleus and the DAT-OFF nucleus.)

PSA
: The PSA for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

RGN
: The allocated pages in the private area of each address space being dumped. This includes the following areas:

Subpools	Storage
0-127, 129-132, 229, 230, 240, 249, 250-252	All storage allocated to these subpools
203-205, 213-215, 223-225, 233-235, 253-255	All storage allocated to the LSQA and ELSQA
236, 237	All storage allocated to the SWA and ESWA

SVC dump does not dump all the obtained private storage in an address space when the RGN option of SDATA is specified. This reduces the number of page faults that occur during SVC dump processing, decreases the time required to take a dump, and reduces the size of the dump.

SVC dump dumps only those obtained pages that have something stored into them. Data-in-virtual pages that have been changed since the last DIV macro (that specified the SAVE service) executed, will also be dumped.

Eliminated from the dump are the pages of storage that are in a freshly-obtained state. Data-in-virtual pages that are in central storage, but have not been changed, are also considered to be in a freshly-obtained state. They will not be dumped.

SQA
: The SQA and ESQA subpools (226, 239, 245, 247, and 248).

SUMDUMP or SUM
: A summary dump is to be included with the SVC dump output. The trace table is included in the nonsummary portion of the dump if this option is specified or used as a default.

  The type of summary dump depends on how you specify the BRANCH and SUSPEND parameters:

  - If you specify BRANCH=YES and SUSPEND=NO, you get a disabled summary dump.
  - If you specify BRANCH=YES and SUSPEND=YES, you get a suspend summary dump.
  - If you specify BRANCH=NO, you get an enabled summary dump.

  For a description of the dump contents, see *OS/390 MVS Diagnosis: Tools and Service Aids*.

SWA The scheduler work area subpools for each address space being dumped (subpools 236 and 237). This includes all storage allocated above and below 16 megabytes.

TRT The system trace table, the GTF trace records, and master trace data if these types of traces are active.

DEFAULTS or DEFS
The following default SDATA options are included in the SVC dump:
- ALLPSA
- SQA
- SUMDUMP
- IO
- Any default SDATA options specified by the CHNGDUMP command when CHNGDUMP is in ADD mode.

Notes:

1. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS is specified.
2. DEFAULTS and NODEFAULTS are mutually exclusive.

NODEFAULTS or NODEFS
The SDATA defaults are NOT included in the SVC dump. Specifying NODEFAULTS reduces the size of an SVC dump by excluding the following default SDATA options:
- ALLPSA
- SQA
- SUMDUMP
- IO
- Any default SDATA options specified by the CHNGDUMP command when CHNGDUMP is in ADD mode.

If a data area relating to an SDATA option is required in the dump, the programmer can code that SDATA option on the SDUMP macro invocation. All keywords and SDATA options are valid when NODEFS is coded.

If you specify NODEFAULTS, the dump still contains some default system areas that are included in all dumps.

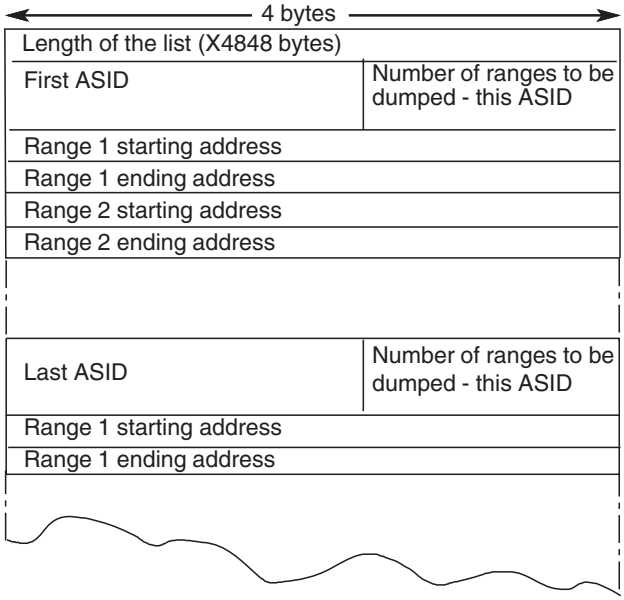
IO The IO data areas are included in the SVC dump.

,STORAGE=*(strt addr,end addr)*
,LIST=*list addr*
,LISTA=*listaddr*

Specifies one or more pairs of starting and ending addresses (STORAGE), a list of starting and ending addresses (LIST), or a list of ASIDs and storage ranges (LISTA). Each starting address must be less than its corresponding ending address.

When LIST or STORAGE is specified, the list must contain an even number of addresses, and each address must occupy one fullword. In the list, the high-order bit of the fullword containing the last ending address of the list must be set to 1; all other high-order bits must be set to 0.

When LISTA is specified, the first fullword of the storage list contains the number of bytes (including the first word) in the list. LISTA specifies a list of ASIDs and storage ranges as follows:



Note: If LISTA or SUBPLST is specified for a scheduled dump request and if the list does not exceed 484 bytes in size, SVC dump will move the list to SVC dump storage. The caller can free or reuse this space on return from SVC dump. No synchronization with SVC dump completion is required. If the list exceeds 484 bytes, SVC dump will not move the list and synchronization with SVC dump completion is required.

,SUBPLST=*subpool id list address*

Specifies a list of ASIDs with associated subpool ids corresponding to subpools of virtual storage that are to be included in the SVC dump.

The first fullword of the list contains the number of bytes (including the first word) in the list. The list can contain a maximum of 200 bytes consisting of unique ASIDs and subpool IDs. If the list contains duplicate ASIDs or subpool IDs, the length can exceed 200 bytes because SDUMP stores the unique subpool IDs in a 200-byte work area.

The structure of the list for each ASID follows:

- The first word contains the ASID in bits 0-15 and the number of subpools associated with this ASID (n) in bits 16-31. If 0 is specified as the ASID, the caller's home ASID is used.
- The next n halfwords contain the subpool IDs (right justified) corresponding to the virtual storage to be included in the SVC dump. The manner in which these subpools are dumped depends on whether they are private or common area subpools.
  - If a private area subpool (related to a TCB) is specified, all virtual storage associated with this subpool, for all TCBs in the specified address space, is dumped.
  - If a common area subpool is specified, all of the virtual storage allocated in the subpool is dumped.

SVC dump does not dump all the obtained storage in an address space if the SUBPLST list keyword for private subpools is coded. This reduces the number of page faults that occur during SVC dump processing and the time required to take a dump. It also reduces the size of dumps on tape or DASD.

For storage that is not related to data-in-virtual, only obtained pages that have something stored into them are dumped. This eliminates the pages of storage that are in a freshly obtained state.

For storage that is related to data-in-virtual, only pages that are in central storage are dumped, as well as pages that have been changed since the last data-in-virtual SAVE operation.

Notes:

1. SVC dump ignores unassigned subpool IDs and ASIDs.
2. If an invalid subpool or ASID (ASID greater than ASVTMAXU) is specified, the caller receives a 233 ABEND and SDUMP processing terminates the dump.
3. If all ASIDs specified in SUBPLST are the current ASID, SUBPLST does not force a scheduled dump. However, if any of the ASIDs are different, a scheduled (or asynchronous) dump results.
4. SDUMP callers executing in key 0 and supervisor state, who request storage from subpool 0 via GETMAIN obtain that storage from subpool 252 instead. Therefore, when these callers want to dump this storage, they must specify subpool 252 rather than subpool 0.

,KEYLIST=*storage key list addr*

Specifies the address of a list of storage keys associated with the virtual storage to be dumped. If the key of a subpool specified in SUBPLST does not match a key in this list, the data in the subpool is not dumped. SUBPLST must be specified if the KEYLIST option is used. If you do not specify KEYLIST, all storage (regardless of key) associated with the requested subpools is included in the dump. Therefore, if you want to dump the storage corresponding to all 16 keys, do not specify this parameter.

The list contains one-byte entries and starts on a halfword boundary. The first byte indicates the length of the list (including this byte). The list has a maximum length of 16 bytes so that up to 15 keys can be specified. Specify each key in the left-most four bits of each byte, except the length byte.

,BUFFER=NO
,BUFFER=YES

Specifies that the contents of the SQA buffer is (YES) or is not (NO) to be included in the dump. (The SQA buffer does not include the SDUMP parameter list or any data pointed to by the parameter list.) Callers who specify BUFFER=YES on the SDUMP macro will obtain a dump of a 4K buffer reserved in the SQA for the callers of SVC dump. You can reserve the buffer by setting the high-order bit of the CVTSDBF field in the communications vector table (CVT). Once you have reserved the buffer, you can fill it with data before issuing SDUMP. Programs that are involved with data that might change before SDUMP can dump it should use this buffer.

The CVTSDBF field of the CVT points to the buffer. Before using the buffer, use compare and swap logic to serialize on the high-order bit of CVTSDBF. If the bit was on (B'1'), the buffer is in use, and you should continue processing as though a dump could not be taken. If the bit was off (B'0'), the bit is set to B'1' by the compare and swap. You can then fill the buffer and issue SDUMP. If the compare and swap sets the CVTSDBF bit, SDUMP resets it for you.

,QUIESCE=YES
,QUIESCE=NO

Specifies that the system is to be set nondispatchable until the contents of the SQA and the CSA are dumped (YES), or that the system is to be left dispatchable (NO). If the SDATA parameter does not specify SQA or CSA, the QUIESCE=YES request is ignored.

Note: Summary dumps (SUMDUMP) for branch entries (BRANCH=YES) always cause the system to be set nondispatchable until the summary dump is written.

,BRANCH=NO
,BRANCH=YES

Specifies that a branch entry is to be used for interfacing with SVC dump to schedule a dump (YES), or that an SVC instruction is to be generated for interfacing with SVC dump (NO).

For BRANCH=YES, MVS produces a scheduled (asynchronous) SVC dump. For BRANCH=NO, the parameters you code to identify storage determine whether MVS produces a scheduled or synchronous SVC dump. MVS produces a scheduled dump when you code BRANCH=NO with one or more of the following:

- ASIDLIST
- ASID=*asid addr*
- TYPE=XMEM or TYPE=XMEME
- LISTA
- LISTD=*list addr*, when the STOKEN represents either an address space other than the primary address space, or a SCOPE=SINGLE data space owned by a program that is not running in the primary address space
- SUBPLST=*subpool id list addr*, when the list of address spaces with associated subpool IDs contains at least one address space other than the primary address space.

You might make different design decisions for your program based on the type of dump MVS produces. See *OS/390 MVS Programming: Authorized Assembler Services Guide* for guidance about designing your program to handle each type of dump. If BRANCH=YES is specified and the caller has not specified at least one of the following keywords: ASID, ASIDLST, TYPE=XMEM, TYPE=XMEME, or LISTA, the dump is scheduled to the home address space. Routines that issue SDUMP with BRANCH=YES must provide a 72-byte save area pointed to by register 13, and must include the CVT mapping macro.

For BRANCH=YES entry by reentrant recovery routines, SDUMP processing moves the data supplied by the following parameters to a system area:

HDR
HDRAD
ID
IDAD
ASIDLIST
STORAGE
LIST
LISTA
SUBPLST
KEYLIST

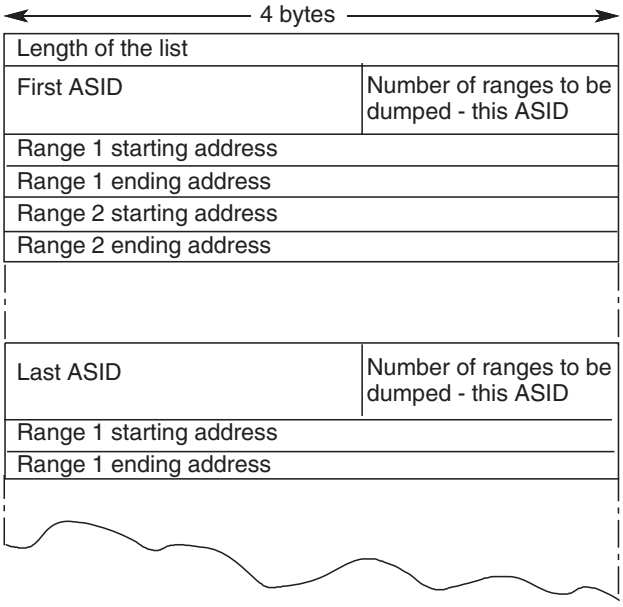
This enables the recovery routine to free its storage on return from SDUMP although the dump has not completed.

,SUSPEND=NO
,SUSPEND=YES
Specifies that a suspend summary dump is requested (YES) or not requested (NO). SUSPEND=YES must be used together with the BRANCH=YES and SDATA=SUMDUMP parameters. This keyword should be used by routines that can experience page faults but that want to save dump information in a virtual storage buffer.

,SUMLIST=*list addr*
,SUMLSTA=*list addr*
Specifies a list of starting and ending addresses of areas to be included in a summary dump (SUMLIST) or specifies a combined list of ASIDs and storage ranges (SUMLSTA). SUMDUMP must be specified as an SDATA parameter and each starting address must be less than its corresponding ending address.

For SUMLIST, the storage list must contain an even number of addresses, and each address must occupy one fullword. In the list, the high-order bit of the fullword containing the last ending address of the list must be set to 1, and all other high-order bits must be set to 0.

For SUMLSTA, the first fullword of the list contains the number of bytes (including the first word) in the list. SUMLSTA specifies a list of ASIDs and storage ranges as follows:



Restriction:

- The maximum number of ASIDs that the combined TYPE=XMEM, TYPE=XMEME, LISTA, ASIDLST, ASID, and SUBPLST parameters can specify is fifteen.

  Note: There is no restriction on the number of ASIDs that the SUMLSTA can specify.

When BRANCH=YES and SUSPEND=NO are also specified, the list must be addressable using the addressability established when SVC dump was entered. The lists themselves and all ranges specified must reference paged-in data. Paged-out data is not dumped by summary dump.

When BRANCH=YES and SUSPEND=YES are also specified, the lists must be addressable using the addressability established when SVC dump was entered. The lists and referenced data can either be in paged in or paged out areas. The maximum amount of summary dump data with this type of dump is 1M.

When BRANCH=NO is also specified, the lists must be addressable in all address spaces in which the dump will be taken (those address spaces specified by ASID, ASIDLST, LISTA, or TYPE=XMEM, TYPE=XMEME, or SUBPLST). The lists and referenced data can be in paged-in or paged-out areas. The maximum amount of summary dump data possible with this type of dump is dependent only on the size of the dump data set.

Each ASID specified with SUMLSTA must represent a valid, swapped-in address space in order for the data to be dumped.

Programming Notes: The total number of distinct ASIDs that can be specified by TYPE=XMEM, TYPE=XMEME, LISTA, ASID, SUBPLST and ASIDLST is fifteen. If more than fifteen are requested, only the first fifteen are processed. There is no restriction on the number of ASIDs specified by the SUMLSTA parameter, nor do SUMLSTA ASIDs contribute toward the fifteen ASID limit.

Return and Reason Codes

The following tables identify return codes and reason codes, tell what each means, and recommend actions that you should take.

Register 15 Return Codes

If BRANCH=NO was specified and no ASIDs other than the current ASID were requested, register 15 contains one of the following hexadecimal return codes when control is returned at the completion of the capture phase:

Figure 52. Return Codes for the SDUMP Macro when BRANCH=NO

Return Code	Meaning and Action
00	**Meaning**: A complete dump was taken. **Action**: None
04	**Meaning**: A partial dump was taken because the dump data set did not have sufficient space. **Action**: Examine the reason code that explains why a partial dump was taken. The reason code is contained in message IEA911E. If you specified the SRB parameter, you can include the IHASDRSN mapping macro to map the reason code information.
08	**Meaning**: The system was unable to take a dump. **Action**: Examine the reason code that explains why no dump was taken (see “Reason Codes for Return Code 08” on page 268).

If BRANCH=YES or any ASID other than the current ASID was requested, register 15 contains one of the following hexadecimal return codes when control is returned after the system has scheduled the dump:

Figure 53. Return Codes for the SDUMP Macro when BRANCH=YES

Return Code	Meaning and Action
00	**Meaning**: A dump was scheduled. **Action**: None
08	**Meaning**: The system was unable to schedule a dump. **Action**: Examine the reason code that explains why no dump was taken (see “Reason Codes for Return Code 08” on page 268).

ECB and SRB Return Codes

If you specify the ECB or SRB parameter without the DCB parameter, the system also returns one of following hexadecimal codes in the ECB or SRB at the completion of the capture phase:

Figure 54. Return Codes for the ECB Parameter and SRB Parameter

Return Code	Meaning and Action
00	**Meaning**: All the requested data was captured and the dump writing phase was successfully initiated. **Action**: None
04	**Meaning**: Some of the requested data could not be captured and one or more partial dump indicators have been set in SDRSN. The dump writing phase was successfully initiated. **Action**: Examine the reason code that explains why a partial dump was taken. The reason code is contained in message IEA911E. If you specified the SRB parameter, you can include the IHASDRSN mapping macro to map the reason code information.
08	**Meaning**: The system was unable to take a dump. **Action**: Examine the reason code that explains why no dump was taken (see “Reason Codes for Return Code 08” on page 268).

If you specify the DCB parameter with the ECB or SRB parameter, the system also returns one of the following hexadecimal codes in the ECB or SRB at the completion of the dump writing phase:

Figure 55. Return Codes for the ECB or SRB Parameter with the DCB Parameter	
Return Code	**Meaning and Action**
00	**Meaning**: All the requested data was captured and then written to the dump data set. **Action**: None
04	**Meaning**: Some of the requested data could not be captured or could not be written to the dump data set. **Action**: Examine the reason code that explains why a partial dump was taken. The reason code is contained in message IEA911E. If you specified the SRB parameter, you can include the IHASDRSN mapping macro to map the reason code information. The reason codes might also be passed to the SRB routine in the SDSTPDRC field of SDSTATUS.
08	**Meaning**: The system was unable to take a dump. **Action**: Examine the reason code that explains why no dump was taken (see “Reason Codes for Return Code 08” on page 268).

Note: The ECB will not be posted unless the return code from SDUMP is 0.

Reason Codes for Return Code 08

When a return code of 08 is received, a hexadecimal reason code is returned. The reason code is in the following locations:

- In the SDWASDRC field of the SDWA if you issued SDUMP in a recovery routine, and the system provided an SDWA.
- In the ECB or register 15 (bits 16-23), provided that the FAILRC parameter is specified.
- In the SDSTATUS field. This field is pointed to by the SRBPARM field that is in the SRB parameter list. The parameter list is passed to SDUMP by using the SRB keyword.

The reason codes are as follows:

Figure 56 (Page 1 of 3). Reason Codes for Return Code 08	
Reason Code	**Meaning and Action**
0	**Meaning**: No SVC dump was requested. **Action**: None
2	**Meaning**: An SVC dump was suppressed because the capture phase of another SVC dump was in progress. **Action**: Wait until the dump in progress has been captured (as identified by message IEA794I) and reissue SDUMP.
3	**Meaning**: An SVC dump was suppressed by a request by the installation (for example: DUMP=NO at IPL or CHNGDUMP SET,NODUMP). **Action**: Issue CHNGDUMP SET,SDUMP or CHNGDUMP RESET,SDUMP and reissue SDUMP.
4	**Meaning**: An SVC dump was suppressed by a SLIP NODUMP command. **Action**: Delete SLIP trap with SLIP DEL command and reissue SDUMP.
5	**Meaning**: An SVC dump was suppressed because a SYS1.DUMP data set was not available. **Action**: If MSGTIME expired, increase MSGTIME limit with CD SET,SDUMP,MSGTIME= command. Make a dump dataset available via the DUMPDS ADD,DSN= and/or DUMPDS CLEAR,DSN= commands and reissue SDUMP.
6	**Meaning**: An SVC dump was suppressed because an I/O error occurred during the initialization of the SYS1.DUMP data set. **Action**: Reissue SDUMP.
8	**Meaning**: An SVC dump was suppressed because an SRB could not be scheduled to activate the dump tasks in the requested address spaces. **Action**: None
9	**Meaning**: An SVC dump was suppressed because a terminating error occurred in SVC dump before the first dump record was written. **Action**: Reissue SDUMP.

Figure 56 (Page 2 of 3). Reason Codes for Return Code 08

Reason Code	Meaning and Action
A	**Meaning**: An SVC dump was suppressed because a status stop SRB condition was detected. **Action**: None
B	**Meaning**: An SVC dump was suppressed by DAE. **Action**: None.
15	**Meaning**: The parameter list address is zero. **Action**: Supply a parameter list address in register 1 and reissue SDUMP.
16	**Meaning**: The parameter list is not a valid SVC or SNAP parameter list. **Action**: Provide the address of a valid SVC dump parameter list in register 1 and reissue SDUMP.
17	**Meaning**: The caller-supplied data set is not supported. **Action**: Supply a dataset with LRECL >= 4160 open with EXCP on a device supported by SVC dump (or use a system dump dataset) and reissue SDUMP.
18	**Meaning**: The start address is greater than or equal to the end address in a storage list. **Action**: Correct the address range that is not valid and reissue SDUMP.
19	**Meaning**: The caller-supplied header is longer than 100 characters. **Action**: Supply a shorter header and reissue SDUMP.
1A	**Meaning**: The caller requested a 4K buffer, but did not reserve it. **Action**: Set the high-order bit of CVTSDBF and reissue SDUMP.
1B	**Meaning**: A storage list overlaps the 4K buffer. **Action**: Move the storage list so that it does not overlap the SVC dump 4K buffer pointed to by CVTSDBF. Reissue SDUMP.
1C	**Meaning**: The caller-supplied DCB is not valid. **Action**: Make sure DCB is open, does not overlap 4K buffer, and represents a tape or DASD dataset, then reissue SDUMP.
1E	**Meaning**: An ASID in the ASID list is syntactically not valid. **Action**: Supply a valid ASID (<= ASVTMAXU) and reissue SDUMP.
22	**Meaning**: The 4K buffer was requested with an SVC dump already in progress. **Action**: Wait until the dump in progress has been captured and reissue SDUMP.
25	**Meaning**: A nonvalid subpool ID was specified in the subpool list. **Action**: Supply a valid subpool id (<= 255) and reissue SDUMP.
28	**Meaning**: Part of the parameter list is inaccessible. **Action**: Make sure the parameter list is addressable from the caller's current address space. Reissue SDUMP.
29	**Meaning**: The caller-supplied DCB is inaccessible. **Action**: Make sure the DCB is addressable from the caller's current address space. Reissue SDUMP.
2A	**Meaning**: The caller-supplied storage list is inaccessible. **Action**: Make sure the storage list is addressable from the caller's current address space. Reissue SDUMP.
2B	**Meaning**: The caller-supplied header data is inaccessible. **Action**: Make sure the header is addressable from the caller's current address space. Reissue SDUMP.
2C	**Meaning**: The caller-supplied ECB is inaccessible. **Action**: Make sure the ECB is addressable from the caller's current address space. Reissue SDUMP.
2D	**Meaning**: The caller's ASID list is inaccessible. **Action**: Make sure the ASID list is addressable from the caller's current address space. Reissue SDUMP.
2E	**Meaning**: The caller's SUMLIST/SUMLSTA is inaccessible. **Action**: Make sure the SUMLIST/SUMLSTA is addressable from the caller's current address space. Reissue SDUMP.
2F	**Meaning**: The caller's SUBPLST list is inaccessible. **Action**: Make sure the SUBPLST is addressable from the caller's current address space. Reissue SDUMP.

Figure 56 (Page 3 of 3). Reason Codes for Return Code 08

Reason Code	Meaning and Action
30	**Meaning**: The caller's KEYLIST is inaccessible. **Action**: Make sure the KEYLIST is addressable from the caller's current address space. Reissue SDUMP.
31	**Meaning**: Copies of the SLIP register and PSW are inaccessible. **Action**: None
32	**Meaning**: The caller-supplied SRB is inaccessible. **Action**: Make sure the SRB is addressable from the caller's current address space. Reissue SDUMP.
33	**Meaning**: The version number in the parameter list is not valid. **Action**: Supply a parameter list with a valid version number and reissue SDUMP.
34	**Meaning**: The caller's LISTD is inaccessible. **Action**: Make sure the LISTD is addressable from the caller's current address space. Reissue SDUMP.
35	**Meaning**: The caller's SUMLISTL is inaccessible. **Action**: Make sure the SUMLISTL is addressable from the caller's current address space. Reissue SDUMP.
36	**Meaning**: The parameter list contains conflicting parameters. **Action**: Remove the conflicting parameters (for example, both ECB and SRB specified) and reissue SDUMP.
37	**Meaning**: The ID is longer than 50 characters. **Action**: Supply a shorter ID and reissue SDUMP.
38	**Meaning**: The ID is not addressable. **Action**: Make sure the ID is addressable from the caller's current address space. Reissue SDUMP.
39	**Meaning**: The PSWREGS area is an incorrect length. **Action**: Correct the length of the PSWREGS area and reissue SDUMP.
3A	**Meaning**: The PSWREGS area is not addressable. **Action**: Make sure the PSWREGS area is addressable from the caller's current address space. Reissue SDUMP.
3B	**Meaning**: The symptom record is not valid. **Action**: Supply a valid symptom record and reissue SDUMP.
3C	**Meaning**: The symptom record is not addressable. **Action**: Make sure the symptom record is addressable from the caller's current address space. Reissue SDUMP.
3D	**Meaning**: The DEB for the caller-supplied DCB is inaccessible. **Action**: Make sure the DEB for the caller-supplied DCB is addressable from the caller's current address space. Reissue SDUMP.
3E	**Meaning**: SVC dump is already using the maximum amount of virtual storage (as determined by the installation, using the MAXSPACE parameter on the CHNGDUMP command) to process other dumps. **Action**: Make a dump dataset available via the DUMPDS ADD,DSN= or DUMPDS CLEAR,DSN= command, reply DELETE to an outstanding IEA793A message, or increase the amount of virtual storage that SDUMP is allowed to use via the CHNGDUMP SET,SDUMP,MAXSPACE= command, then reissue SDUMP.
46	**Meaning**: SVC dump stopped the dump because the system resources manager (SRM) detected a critical shortage of auxiliary storage. **Action**: See the system programmer response for message IRA201E to determine how to relieve the shortage. Then reissue the SDUMP macro.
FF	**Meaning**: An SVC dump was suppressed for some other unspecified reason. **Action**: None

Example 1

This example shows how SVC dump can be branch entered to initiate a dump in an address space by callers who cannot issue an SVC. Areas to be dumped are requested via three parameters (BUFFER, SDATA, and STORAGE). The dump has the title indicated in the HDR parameter, the caller requests to be notified of the completion of the scheduled dump via the ECB parameter, and the dump is going to a private data set (indicated by the DCB option).

```
SDUMP HDR='USER DATA FOR TEST A',DCB=TESTADCB,BUFFER=YES,  X
  ASID=TSTAASID,ECB=(8),QUIESCE=YES,BRANCH=YES,           X
  STORAGE=(A,B,C,D,(9),E),SDATA=(ALLPSA,SQA,LSQA)
```

Example 2

This example shows how SVC dump can be invoked via a branch entry to initiate a dump of several address spaces by callers who cannot issue an SVC. Areas to be dumped are requested via four parameters (BUFFER, SDATA, LIST, and SUMLIST). The address spaces to be dumped are described by the ASIDLST parameter. Note that areas specified by SUMLIST only apply to the primary address space. The LIST addressed by the LIST keyword must be addressable from any address space. The dump has the title indicated in the HDR parameter, and the caller requests to be notified of the completion of the scheduled dump via the ECB parameter.

```
SDUMP HDR='USER DATA FOR TEST B',                         X
  BUFFER=YES,ASIDLST=TSTALIST,ECB=(8),                    X
  QUIESCE=YES,BRANCH=YES,LIST=(9),                        X
  SDATA=(ALLPSA,NUC,SQA,SUMDUMP),                         X
  SUMLIST=TSTSLIST
 .
 .
 .
TSTALIST DC X'0000000A800B'
TSTSLIST DC X'0000000080400000'
```

SDUMP—List Form

Use the list form of the SDUMP 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 SDUMP must be equal to the maximum number of addresses specified in the execute form of the macro.

Syntax

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

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: From 1 to 100 characters. *dump title addr*: A-type address.
,DCB=*dcb addr*	*dcb addr*: A-type address.
,PLISTVER=1	*decimal digit 1*: Use up to 68-byte parameter list

,PLISTVER=2	*decimal digit 2*: Use 128-byte parameter list. **Default**: PLISTVER=1, unless you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFS, SDATA=NODEFS, or SDATA=IO, in which case the default is PLISTVER=2.
,SYMREC=*symrec addr*	*symrec addr*: RX-type address, or register (2) - (12).
,ID='*identifier*' ,IDAD=*identifier addr*	*identifier*: From 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,SDATA=*(sdata options)*	*sdata options*: Any combination of the following, separated by commas: ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing the SDUMP macro causes ALLPSA, SQA, IO, and SUMDUMP storage areas to be dumped unless excluded by NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP. 2. The PSA and IO options are not required unless NODEFAULTS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS has been specified.
,STORAGE=*(strt addr,end addr)* ,LIST=*list addr* ,LISTA=*list addr*	*strt addr*: A-type address. *end addr*: A-type address. *list addr*: A-type address **Note:** Specify one or more pairs of addresses, separated by commas.
,SUBPLST=*subpool id list addr*	*subpool id list addr*: A-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr*: A-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default**: BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default**: QUIESCE=YES
,TYPE=*(type code)*	*type code*: Any combination of the following, separated by commas: XMEM or XMEME, NOLOCAL.

,MF=L

Parameters

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

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

Note: If SYMREC, ID, IDAD, PSWREGS, SDATA=NODEFS, SDATA=DEFS or SDATA=IO is not used on the list form of the macro, but is coded on the execute form, use PLISTVER=2 when specifying MF=L to generate a 128-byte parameter list.

SDUMP—Execute Form

A remote control program parameter list is referred to and can be modified by the execute form of the SDUMP macro.

If you code one or more of the SDATA parameters on the execute form of the macro, any SDATA parameters coded on the list form are lost.

Syntax

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

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: From 1 to 100 characters. *dump title addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,ASID=*ASID addr* ,ASIDLST=*list addr*	*ASID addr*: A-type address, or register (2) - (12). *list addr*: RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code*: Any of the following, separated by commas: XMEM, XMEME, NOLOCAL, FAILRC **Note**: XMEM and XMEME are mutually exclusive codes.
,PLISTVER=1 ,PLISTVER=2	*decimal digit 1*: Use up to a 68-byte parameter list. *decimal digit 2*: Use 128-byte parameter list. **Default**: PLISTVER=1, unless you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFS, SDATA=NODEFS, or SDATA=IO, in which case the default is PLISTVER=2.
,SYMREC=*symrec addr*	*symrec addr*: RX-type address, or register (2) - (12).
,ID='*identifier*' ,IDAD=*identifier addr*.	*identifier*: From 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,ECB=(*ecb addr*) ,SRB=(*srb addr*)	*ecb addr*: A-type address, or register (2) - (12). *srb addr*: A-type address, or register (2) - (12). **Notes:** 1. SVC dump posts the ECB at the completion of the capture phase unless the DCB parameter is specified with the ECB parameter. If you specify both the DCB and ECB parameters, the ECB is posted at the completion of the writing phase. 2. SVC dump schedules the SRB at the completion of the capture phase unless the DCB parameter is specified with the SRB parameter. If you specify both the DCB and SRB parameters, the SRB is scheduled at the completion of the writing phase.
,SDATA=*(sdata options)*	*sdata options*: Any combination of the following, separated by commas:

	ALLNUC, ALLPSA, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing SDUMP causes ALLPSA, SQA, IO, and SUMDUMP storage areas to be dumped unless excluded by NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP. 2. The PSA and IO options are not required unless NODEFAULTS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS has been specified.
,STORAGE=*(strt addr,end addr)* ,LIST=*list addr* ,LISTA=*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**: Specify one or more pairs of addresses, separated by commas.
,SUBPLST=*subpool id list addr*	*subpool id list addr*: RX-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr*: RX-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default**: BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default**: QUIESCE=YES
,BRANCH=NO ,BRANCH=YES	**Default**: BRANCH=NO **Note**: If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default**: SUSPEND=NO
,SUMLIST=*list addr* ,SUMLSTA=*list addr*	*list addr*: RX-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 SDUMP macro, with the following exception:

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

Example 1

The execute form is used to change SDATA areas, BUFFER, and QUIESCE options in the SDUMP parameter list. The list form of SDUMP was previously used to create the basic SDUMP parameter list located by register 1.

```
SDUMP SDATA=(SQA,LPA),BUFFER=NO,QUIESCE=NO,MF=(E,(1))
```

Example 2

Operation: This example shows a dump request from SUBSYSTEM1. This dump will be suppressed if the symptoms in the symptom record match a previous dump's symptoms, and if the installation has enabled dump suppression. The dump does not include the SDATA options specified on CHNGDUMP or the ALLPSA or SQA data areas. The dump does include the IO data areas and a summary dump which contains PSW/register data.

```
SDUMP ID='SUBSYSTEM1',SYMREC=(8),SDATA=(NODEFS,IO),PSWREGS=(9)
```

SDUMPX — Dump Virtual Storage

Description

The SDUMPX macro invokes SVC dump to provide a fast unformatted dump of virtual storage or coupling facility structure information to a data set. Programs that run in either primary or access register (AR) mode can use SDUMPX. SDUMPX is similar to SDUMP, except that SDUMPX can generate code and addresses that are appropriate for AR mode, whereas SDUMP cannot. All parameters on SDUMP are valid for SDUMPX. Parameters available only on SDUMPX are: LISTD; SUMLSTL; STRLIST; COUPLE, WLM, and XESDATA options on SDATA; options on ECB and SRB; REMOTE; INTOKEN; PROBDESC; JOBLIST; DSPLIST; and PLISTVER=3.

To dump data space storage, issue SDUMPX and specify either the DSPLIST, LISTD or SUMLSTL parameters.

There are two phases in SVC dump processing:

- The capture phase, in which all the volatile data for the dump is copied into DUMPSRV data spaces.
- The writing phase, in which the data is written to the dump data set.

The caller can initiate an SVC dump in an address space other than the primary. A branch entry is available for callers who wish a dump of their own or another address space, but cannot issue an SVC.

When you request a dump of virtual storage, the combination of parameters you code determines whether MVS produces either a **scheduled** (asynchronous) or a **synchronous** SVC dump. You might make different design decisions for your program based on the type of dump that MVS produces. Read the information about dumping virtual storage in *OS/390 MVS Programming: Authorized Assembler Services Guide* for the parameter combinations that produce each type of dump, and for guidance about designing your program to handle each type.

For information about how to select this macro for an MVS/SP version other than the current version, see “Selecting the Macro Level” on page 7.

Except for the data control block (DCB) parameter, all input parameters to this macro can reside in storage above 16 megabytes if the caller is executing in 31-bit addressing mode.

You can produce reentrant code using the standard form of SDUMPX if you do not specify parameters other than the following:

SDATA
TYPE
HDR
ID
BRANCH
SUSPEND
QUIESCE
BUFFER
PLISTVER

SVC dump allows programs in page-protected storage (such as the nucleus, PLPA, and MLPA) to issue the standard form of the SDUMPX macro without causing a protection exception. However, IBM recommends using the list and execute forms of SDUMPX for programs in page-protected storage.

© Copyright IBM Corp. 1988, 2000

Wildcards

You can use wildcards to identify multiple names. On an SDUMPX macro, you can specify wildcards in job names, data space names, and system names. The parameter descriptions tell you when you can use wildcards. The wildcards are:

Wildcards	Meaning
*	Zero or more characters, up to the maximum length of the string. An * can start the string, end it, appear in the middle, or appear in several places in the string. A single * for the name indicates that all job names, data space names, or system names will match.
?	One character. One or more ? can start the string, end it, appear in the middle, or appear in several places in the string. A single ? indicates all names consisting of one character.

Note: You can mix wildcards in any combination.

Examples are:

- *A* specifies all names that contain an A, including the name A.
- *A*B specifies all names that contain an A followed by and ending with a B, with or without any intervening characters. The name can have characters before the A.
- ?A? specifies all 3-character names with an A as the second character.
- ?A?B specifies all 4-character names with A as the second character and B as the fourth character.
- ?A* specifies all names of 2 or more characters with A as the second character.

Environment

The requirements for the caller with **BRANCH=NO** are:

Minimum authorization:	Any one or more of the following: Supervisor state PSW keys 0 - 7 APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters and all areas the parameter list points to, except the DCB, ECB, and SRB, must be in an address/data space that is addressable from the current address space. The DCB must be addressable in the home address space. The ECB and SRB must be addressable from each address space included in the dump. The SRB must be addressable from the address space in which it will run.

The requirements for the caller with **BRANCH=YES** are one of the following:

- Be in SRB mode
- Hold any lock
- Have an enabled-unlocked-task FRR on the FRR stack

Assuming that one of the above conditions has been satisfied, the other requirements for callers with BRANCH=YES are:

Minimum authorization:	All of the following: Supervisor state PSW key 0
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 and all areas the parameter list points to, except the DCB, ECB, and SRB, must be in an address/data space that is addressable from the current address space. The DCB must be addressable in the home address space. The ECB and SRB must be addressable from each address space included in the dump. The SRB must be addressable from the address space in which it will run.

Programming Requirements

- For AR mode callers:
  - Issue the SYSSTATE ASCENV=AR macro before SDUMPX. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.
  - The parameter list address must be qualified by an ALET of zero.
- To generate reentrant code, code the list and execute forms of the SDUMPX macro. Because the execute form of the macro is dependent on the length determined by the list form, the list form must appear before the execute form in a reentrant program.

  Callers can determine the length of the parameter list by using the following programming technique to calculate the amount of storage needed for only those options specified for the SDUMPX macro:

```
SDMPXBEG SDUMPX SDATA=(SUM),SUMLIST=SLIST,MF=L
SDMPXEND EQU   *
SDMPXLEN DC    A(SDMPXEND-SDMPXBEG)
```

Callers that issue SDUMPX with BRANCH=YES must include the CVT mapping macro.

Restrictions

None.

Input Register Information

If BRANCH=YES is specified, before issuing the SDUMPX macro, the caller must ensure that the following general purpose register (GPR) contains the specified information.

Register	Contents
13	The address of a 72-byte save area

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

Output Register Information

When control returns to the caller, the GPRs contain:

Register	Contents
0-1	Used as work registers by the system
2-13	Unchanged
14	Used as a work register by the system
15	Return code in bits 24-31. If the return code is X'08', and you specified the FAILRC parameter, GPR 15 also contains a reason code in bits 16-23.

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 SDUMPX macro is written as follows:

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: From 1 to 100 characters. *dump title addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,ASID=*ASID addr* ,ASIDLST=*list addr*	*ASID addr*: A-type address, or register (2) - (12). *list addr*: RX-type address, or register (2) - (12).
,JOBLIST=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,DSPLIST=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code*: Any of the following, separated by commas: XMEM, XMEME, NOLOCAL, FAILRC **Note**: XMEM and XMEME are mutually exclusive codes.
,PLISTVER=1 ,PLISTVER=2 ,PLISTVER=3	*decimal digit 1*: Use up to 112-byte parameter list. *decimal digit 2*: Use 128-byte parameter list. *decimal digit 3*: Use 184-byte parameter list. **Default**: PLISTVER=2, if you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFS, SDATA=NODEFS, or SDATA=IO. PLISTVER=3, if you specify STRLIST, REMOTE, INTOKEN, DSPLIST, JOBLIST, or PROBDESC. Otherwise, the default is PLISTVER=1.
,SYMREC=*symrec addr*	*symrec addr*: RX-type address, or register (2) - (12).
,ID='*identifier*' ,IDAD=*identifier addr*	*identifier*: From 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,INTOKEN=*token addr*	*token addr*: RX-type address, or register (2) - (12).
,REMOTE=*area addr*	*area addr*: RX-type address, or register (2) - (12).
,STRLIST=*structure list addr*	*structure list addr*: RX-type address or register (2) - (12). Use the IHABLDP macro to build the structure list address.

,PSWREGS=*parm list addr*

parm list addr: RX-type address, or register (2) - (12).

,ECB=(*ecb addr*)
,ECB=(*ecb addr*,CAPTURE)
,ECB=(*ecb addr*,WRITE)

ecb addr: A-type address, or register (2) - (12).

,SRB=(*srb addr*)
,SRB=(*srb addr*,CAPTURE)

srb addr: A-type address, or register (2) - (12).

Notes:

1. If you code ECB=(*ecb addr*), without specifying CAPTURE or WRITE, SVC dump posts the ECB at the completion of the capture phase unless you also specify the DCB parameter. If you specify both the ECB and DCB parameters, the ECB is posted at the completion of the writing phase.
2. If you code SRB=(*srb addr*), without specifying CAPTURE or WRITE, SVC dump schedules the SRB at the completion of the capture phase unless you also specify the DCB parameter. If you specify both the SRB and DCB parameters, the SRB is scheduled at the completion of the writing phase.

,SRB=(*ecb addr*,WRITE)

,SDATA=*(sdata options)*

sdata options: Any combination of the following, separated by commas:
ALLNUC, ALLPSA, COUPLE, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT, XESDATA, DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO

Notes:

1. Executing SDUMPX causes ALLPSA, SQA, IO, and SUMDUMP storage areas to be dumped unless excluded by the NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP parameter.
2. The PSA and IO options are not required unless NODEFS is specified, because they are dumped as a default in all SVC dumps.
3. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS has been specified.

,PROBDESC=*area addr*

area addr: RX-type address, or register (2) - (12).

,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*
,LISTA=*list addr*
,LISTD=*list addr*

list addr: A-type address, or register (2) - (12).

Note: Specify one or more pairs of addresses, separated by commas.

,SUBPLST=*subpool id list addr*

subpool id list addr: RX-type address, or register (2) - (12).

,KEYLIST=*storage key list addr*

storage key list addr: RX-type address, or register (2) - (12).

Note: KEYLIST cannot be specified without SUBPLST.

,BUFFER=NO
,BUFFER=YES

Default: BUFFER=NO

,QUIESCE=YES
,QUIESCE=NO

Default: QUIESCE=YES

,BRANCH=NO
,BRANCH=YES

Default: BRANCH=NO
Note: If BRANCH=YES is specified, ASID or ASIDLST must also be specified.

,SUSPEND=NO
,SUSPEND=YES

Default: SUSPEND=NO

,SUMLIST=*list addr* ,SUMLSTA=*list addr* ,SUMLSTL=*list addr*	*list addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

HDR=*'dump title'*
HDRAD=*dump title addr*

Specifies the title or address of the title to be used for the dump. If HDR is specified, the title must be 1-100 characters enclosed in apostrophes, although the apostrophes do not appear in the actual title. If HDRAD is specified, the first byte at the indicated address specifies the length of the title in bytes.

If the length of the title is greater than 100, SVC dump issues an abend with a completion code of X'233', reason code X'14', then returns to the caller with a return code of 8. If the length of the title is zero, SVC dump continues processing as if the HDR or HDRAD parameter was not specified.

If these keywords are specified with BRANCH=YES or ASID/ASIDLST (that is, to cause a scheduled dump), the system moves the title to SVC dump storage before it returns control to the caller. There is no requirement to synchronize with the completion of the dump.

,DCB=*dcb addr*

Specifies the address of a previously opened data control block (DCB) for the data set that is to contain the dump. If this parameter is omitted, one of the SYS1.DUMP data sets is used. When you specify the DCB parameter, the dump contains data from only the requestor's home address space. The DCB must be addressable from the home address space. The control blocks built by OPEN must also be addressable from the home address space. The DCB must support EXCP. You must specify the following parameters on the DCB macro: RECFM=FB, LRECL=4160, and BLKSIZE=4160.

The DCB must reference device types supported by SVC dump. Eligible device types are unlabeled 9-track 2400-series tape devices (or tape devices compatible with the 2400-series) and any direct access devices supported by the system that have a track size of at least 4160 bytes. (4160 bytes equals 1 SVC dump output record.) SVC dump does not support secondary extents on DCB data sets.

SVC dump does not close the dump data set. Use the CLOSE macro to close the data set and cause an end-of-file mark or a tape mark to be placed after the dump data. SVC dump sets up the DCB so that CLOSE works correctly and positions the end-of-file mark or tape mark at the correct place on the data set. For tape data sets, you can write a tape mark to separate multiple dumps without using the CLOSE macro.

Because it is the caller's responsibility to close the dump data set and the data set may be closed only after all the data has been written to it, the caller needs to receive notification when the dump writing phase is complete. Therefore, if you specify the DCB parameter with the ECB parameter, the system posts the ECB at the completion of the dump writing phase, no matter what has been specified on the ECB parameter. The ECB parameter is required when a DCB is provided for scheduled dumps. If an ECB is not provided with the DCB for a synchronous dump, SVC dump returns to the caller at the completion of the dump writing phase. See *OS/390 MVS Programming: Authorized Assembler Services Guide* for descriptions of scheduled and synchronous dumps.

,ASID=*ASID addr*
,ASIDLST=*list addr*

Specifies the address of a halfword or a list of halfwords containing the hexadecimal address space identifier of an address space to be dumped. If register notation is used, the low-order halfword of the register contains the address space identifier of the address space to be dumped. If both parameters are omitted, the primary address space is dumped. If 0 is specified for the address space identifier, a dump is scheduled

for the home address space of the issuer of the SDUMPX macro. No private area storage is included in the dump for the specified address space if either of the following events occurred:

- No SDATA parameters were specified that apply to the private area of the requested address space.
- The CHNGDUMP operator command was used to set an overriding parameter in the system dump options list that limits SVC dumps to areas outside of the private area.

The ASID list can contain a maximum of 15 address space identifiers. The high-order bit of the halfword containing the last identifier of the list must be set to 1, and all other high-order bits must be set to 0.

If the combined address spaces from the following exceed 15, the system dumps the first 15.

- Specified by the ASID or ASIDLST parameter
- Associated with the jobs specified in the JOBLIST parameter
- Associated with each data space specified in the DSPLIST or LISTD parameter when the data space was created by a DSPSERV CREATE macro with SCOPE=SINGLE
- Associated with the address ranges specified in the LISTD parameter

Wildcards used in the parameters can result in multiple address spaces.

,JOBLIST=*list addr*

Specifies the address of an area that identifies jobs to be dumped. If register notation is used, the register contains the address of the area. If the parameter is omitted, the current job is dumped.

The area must be in common storage. The area consists of:

- **A 4-byte header**, which indicates the total length of the area. The length includes the 4 bytes of the header.

  If the area exceeds 256 bytes, the caller must not free the area until data capture for the dump is completed. For an asynchronous dump, use an ECB or SRB parameter to be notified when data capture is complete or use a SUSPEND parameter to suspend other processing until data capture is completed. (For a synchronous dump on the local system, you do not have to worry about freeing the area too soon because the system does not return control to the caller until processing completes.)

  If the length is less than 4 bytes, the system ignores the JOBLIST parameter.
- **One or more 8-character job names** for jobs to be included in the dump. Left justify each name in its 8-character field; pad it on the right with blanks, if needed. A job name can be specified with wildcards. See “Wildcards” on page 278.

You can specify a maximum of 15 job names. See the ASID parameter for the limit on address spaces that can be specified.

,DSPLIST=*list addr*

Specifies the address of an area that identifies 256 data spaces to be dumped. If wildcard specifications are used, only 256 of the data spaces that match can be dumped. Also, each data space reduces the number of LISTD entries available by one. If register notation is used, the register contains the address of the area. If the parameter is omitted, data spaces are not dumped.

The area must be in common storage. The area consists of:

- **A 4-byte header**, which indicates the total length of the area. The length includes the 4 bytes of the header.

If the area exceeds 512 bytes, the caller must not free the area until data capture for the dump is completed. For an asynchronous dump, use an ECB or SRB parameter to be notified when data capture is complete or use a SUSPEND parameter to suspend other processing until data capture is completed. (For a synchronous dump on the local system, you do not have to worry about freeing the area too soon because the system does not return control to the caller until processing completes.)

If the length is less than 4 bytes, the system ignores the DSPLIST parameter.

- **One or more 16-byte data space identifiers** for data spaces to be included in the dump. An identifier is an ASID or jobname in the left 8 bytes and the data space name in the right 8 bytes:

  ASID
  An 8-byte field containing an explicit hexadecimal address space identifier in bytes 7 and 8. Bytes 1 through 6 must be hexadecimal zeros.

  jobname
  A 1 to 8 character job name left-justified in the 8-byte field. Pad it on the right with blanks, if needed. The job name can be specified with wildcards. See "Wildcards" on page 278.

  data space name
  The 1 to 8 character name associated with the data space at its creation. Left justified in the 8-byte field. Padded on the right with blanks if needed. The data space name can be specified with wildcards. See "Wildcards" on page 278.

See the ASID parameter for the limit on address spaces that can be specified.

,TYPE=XMEM
,TYPE=XMEME
,TYPE=NOLOCAL
,TYPE=FAILRC

Specifies that the caller's cross memory mode determines the address spaces to dump (XMEM or XMEME) or that the caller cannot allow SDUMPX to obtain a local lock (NOLOCAL) or that SVC dump should return a reason code with the return code to the DUMP command processor when the requested dump was not taken (FAILRC).

XMEM requests SVC dump to use the caller's cross memory mode at the time the SDUMPX macro is executed.

XMEME requests SVC dump to use the caller's cross memory mode at the time of the error for which the dump is being taken.

The home address space is dumped for both keywords. The relevant primary and secondary address spaces are also dumped if they are unique. If a cross memory local lock was held, the address space whose local lock is held is also dumped.

NOLOCAL indicates that the caller is in an environment where SDUMPX cannot hold a local lock. This option has meaning only when BRANCH=YES is specified and the caller is enabled and unlocked (for example, the caller has an enabled unlocked task FRR established or is in SRB or cross memory mode).

FAILRC requests that the caller receive special information from SVC dump whenever the dump fails. Some information is already placed in SDWASDRC as a result of the SVC dump failure. When the caller receives control again after a dump failure (return code 8) and the caller has specified TYPE=FAILRC, the reason code is combined with the return code and passed to the caller in either register 15 or the ECB, or through the IHASDST mapping macro if the SRBPARM area was provided for an SRB. The reason code is in bits 16-23; the return code is in bits 24-31. When the return code is in the ECB, the POST flag is set on. SDUMPX passes back a return code in register 15 and places the reason code in the SDWA. The reason code explains why the dump failed.

,PLISTVER=1
,PLISTVER=2
,PLISTVER=3

Specifies the length of the parameter list used:

- For PLISTVER=1, the length is up to 112 bytes
- For PLISTVER=2, the length is 128 bytes
- For PLISTVER=3, the length is 184 bytes.

Defaults are as follows:

- When you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFS, SDATA=NODEFS, or SDATA=IO, the default is PLISTVER=2.
- When you specify STRLIST, REMOTE, INTOKEN, PROBDESC, JOBLIST, and DSPLIST, the default is PLISTVER=3.
- For other specifications, the default is PLISTVER=1.

,SYMREC=*symrec addr*

Specifies the address of a valid symptom record for DAE to use for dump suppression. DAE suppresses the SVC dump if the primary symptom string found in the symptom record matches previously known symptoms, and, suppression has been enabled by the installation.

The caller must build the symptom record and fill in at least the 'SR' identifier and the primary symptom string, which should uniquely identify the error.

SVC dump issues an abend with a completion code of X'233', reason code X'9C', then returns to the caller with a return code of 8 if the symptom record identifier is not 'SR', if the offset and length of the primary symptom string are not initialized, or if the first byte of the symptom record and the last byte of the secondary symptom string are not addressable.

SVC dump does not include the symptom record in the dump. The caller can use the SUMLIST keyword to include the symptom record in the dump.

See the dump analysis and elimination (DAE) section in *OS/390 MVS Programming: Authorized Assembler Services Guide* for more information on symptom strings and how to build them.

The ADSR macro maps the symptom record. See *OS/390 MVS Data Areas, Vol 1 (ABEP-DALT)* for a macro mapping of the ADSR.

,ID=*'identifier'*
,IDAD=*identifier addr*

Specifies an identifier that is included in dump message IEA911E, which is issued at the completion of the dump. The identifier must be from one to 50 printable characters. If ID is specified, the identifier must be enclosed in apostrophes, although the apostrophes do not appear in the actual identifier. If IDAD is specified, the first byte at the indicated address specifies the length of the identifier in bytes. If the length of the identifier is greater than 50, SVC dump issues an abend with a completion code of X'233', reason code X'8C', then returns to the caller with a return code of 8. If the length of the identifier is zero, SVC dump continues processing as if the ID or IDAD parameter was not specified.

,INTOKEN=*token addr*

Specifies the address of a 32-byte area that contains an incident token previously built by an IEAINTKN macro. If register notation is used, the register contains the address of the area.

The system associates the token with the SVC dump on this system and with any SVC dumps requested by the REMOTE parameter on other sysplex systems. If the parameter is omitted, the system generates an incident token.

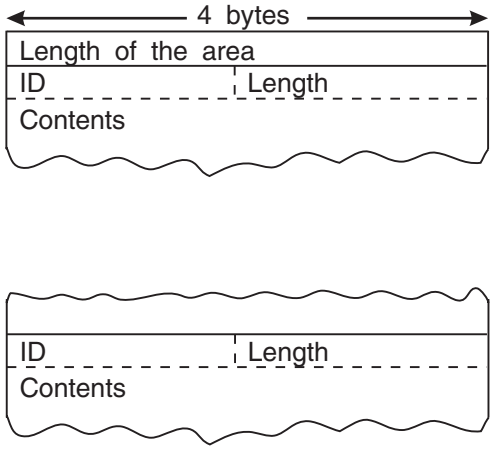
,REMOTE=*area addr*

Specifies the address of an area that identifies other systems in the sysplex to be dumped by this SDUMPX macro. If register notation is used, the register contains the address of the area. If the parameter is omitted, only the current system is dumped.

The area is mapped by the IHASDRMT mapping macro and must be in common storage. Through IHASDRMT, you can identify the systems to be dumped and specify the content of the dumps on individual systems. The area consists of:

- **A 4-byte header**, which indicates the total length of the area. The length includes the 4 bytes of the header.
- **ID, length, contents entries**. Each entry consists of:
  - ID: A 2-byte field, whose value identifies the type of the contents. The values are declared by the constants with names beginning with SDRMT_IDCON in the IHASDRMT mapping.
  - Length: A 2-byte field that gives the length of the contents portion. The length includes the 2 bytes of this length field and the 2 bytes of the ID field.
  - Contents: A variable field that gives the contents identified in the ID field. The contents you can specify are the system names, job names, XCF group and member names, data space names, address space identifiers, SDATA options, storage ranges, subpools, and keys. Within the contents, you can specify items explicitly or, for the following, use wildcards.
    - System name
    - Job name
    - XCF group name
    - XCF member name
    - Data space name and its qualifying job name

    See “Wildcards” on page 278.



If the area exceeds 1024 bytes, the caller must not free the area until data capture for the dump is completed. For an asynchronous dump, use an ECB or SRB parameter to be notified when data capture is complete or use a SUSPEND parameter to suspend other processing until data capture is completed. (For a synchronous dump on the local system, you do not have to worry about freeing the area too soon because the system does not return control to the caller until processing completes.)

If the length of the area is less than 4 bytes, the system ignores the REMOTE parameter. If the length of any of its entries is less than 4 bytes, the dump request returns with an error return code.

The dumps requested through REMOTE are affected by other parameters on the SDUMPX macro: LISTD, SDATA, ASIDLST, JOBLIST, DSPLIST, SUBPLST, and KEYLIST; you can specify that the values of these parameters be copied for the dumps requested through REMOTE. The dumps requested through REMOTE do not contain information for the LIST and LISTA parameters.

,STRLIST=*structure list addr*

Specifies the address of an area that contains a dump parameter list. The list can contain lists of structures, ranges, and options to be dumped from a coupling facility. Use the IHABLDP macro to build this list for input to SDUMPX.

The size of the structure list can range from a minimum of 72 bytes to a maximum of 8K.

,PSWREGS=*list addr*

Specifies a PSW or register area to be passed to SVC dump. This area may contain a PSW, control registers 3 and 4, all the general purpose registers (GPRs), and all access registers (ARs). When PSWREGS is specified, SVC dump includes the following information in the summary dump portion of the dump:

- The PSWREGS parameter list
- If the PSW is provided, 4K of storage before and 4K after the PSW address from the primary address space.
- 4K of storage before and 4K of storage after each of the GPRs from the primary and secondary address spaces.
- If the ARs are provided, they qualify the addresses of the area that includes the 4K of storage before and 4K of storage after each of the GPRs. GPRs will be used to locate storage; ARs (if provided along with a PSW in AR mode) will be used to identify the source address space or data space.

Note: If the control registers are provided, they will be used to determine the primary and secondary address spaces. If no control registers are provided, then the storage will be dumped from the caller's primary and secondary address spaces.

The PSWREGS parameter allows programs running in a nonabend environment, where there is no SDWA, to request SVC dump and dump suppression services similar to those available in an abend environment, where an SDWA is present.

The parameter list for the PSWREGS parameter must reside in the address space currently addressable by SVC dump. The caller must provide at least the length and the mask field. Each bit in the mask refers to a data area. If a bit is set, SVC dump expects that the data area is supplied. If a mask bit is off and any lower-order mask bit is on, the corresponding storage area must be included in the parameter list. If a mask bit is off, but no lower-order mask bits are set, then the storage area need not be supplied.

The following diagram describes the parameter list:

Figure 57 (Page 1 of 2). PSWREGS Parameter List

Offset in Hex	Length	Field Description
00	2	The total length of the PSWREGS parameter list
02	2	Bit mask describing data areas included in the PSW/register area
	1...	Bit 1: On - The PSW is included in the PSW/register area
	.1..	Bit 2: On - Control registers 3 and 4 are included in the PSW/register area
	..1.	Bit 3: On - General purpose registers are included in the PSW/register area
	...1	Bit 4: On - ARs are included in the PSW/register area.
		Bits 5 - 16: Initialize these bits to zero.

Figure 57 (Page 2 of 2). PSWREGS Parameter List

Offset in Hex	Length	Field Description
04	8	PSW: Data only supplied if the PSW mask bit is set
0C	8	Control registers 3 and 4: Data only supplied if mask bit is set.
14	64	General purpose registers 0 - 15: Data only supplied if mask bit is set.
54	64	ARs 0 - 15: Data only supplied if mask bit is set.

,ECB=(*ecb addr***)**
,ECB=(*ecb addr***,CAPTURE)**
,ECB=(*ecb addr***,WRITE)**
,SRB=(*srb addr***)**
,SRB=(*srb addr***,CAPTURE)**
,SRB=(*srb addr***,WRITE)**

Specifies how the system should synchronize your program with dump processing.

ECB specifies that the system should post the event control block (ECB) indicated by *ecb addr*. For ECB=*ecb addr* and ECB=(*ecb addr*,CAPTURE), the system posts the ECB at the completion of the capture phase unless you have also specified the DCB parameter. If you specify both the DCB and ECB parameters, the system posts the ECB at the completion of the writing phase, no matter what has been specified on the ECB parameter. For ECB=(*ecb addr*,WRITE), the system posts the ECB at the completion of the dump writing phase. If the capture phase is not successful, the system posts the ECB at the completion of SVC dump processing.

If an A-type address is specified, *ecb addr* specifies the address of a fullword containing the address of an ECB that is posted on completion of a scheduled dump. If a register operand is used, the register must contain the actual address of the ECB. If this parameter is omitted, the caller is not notified of the completion of the capture phase. The fullword and the ECB must be addressable from the home address space. The fullword address that points to the ECB must be a 24-bit or 31-bit address.

SRB specifies that the system should schedule the service request block (SRB) indicated by *srb addr*. For SRB=*srb addr* and SRB=(*srb addr*,CAPTURE), the system schedules the SRB at the completion of the capture phase unless you have also specified the DCB parameter. If you specify both the DCB and SRB parameters, the system schedules the SRB at the completion of the writing phase, no matter what has been specified on the SRB parameter. For SRB=(*srb addr*,WRITE), the system schedules the SRB at the completion of the dump writing phase. If the capture phase is not successful, the system schedules the SRB at the completion of SVC dump processing.

When the caller builds the SRB, the caller may pass the address of a status area in the SRBPARM field. This status area, SDSTATUS, is mapped by IHASDST. SVC dump passes information about the dump to the SRB routine by means of this status area. If SVC dump schedules the SRB at the completion of the capture phase, the name of the dump data set is not passed to the caller.

,SDATA=*(sdata options)*

Specifies the system is to dump the following:

Option	Data to be Dumped
ALLNUC	The DAT-ON and DAT-OFF nuclei. The read-only (page-protected) area of the nucleus and the DAT-OFF nucleus is not included in the dump unless this keyword is specified.
ALLPSA	All of the prefixed storage areas (PSAs) in the system.
COUPLE	Cross-system coupling facility (XCF) information
CSA	The CSA and ECSA subpools (subpools 227, 228, 231, and 241).

GRSQ Global resource serialization control blocks.

LPA The active link pack area modules and SVCs for each address space being dumped.

LSQA The LSQA and ELSQA for each address space being dumped (subpools 203-205, 213-215, 223-225, 233-235, and 253-255.)

NOALLPSA or NOALL
The PSA for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

NOSQA The system queue area is not dumped.

NOSUMDUMP or NOSUM
A summary dump is not included in the SVC dump.

NUC The non-page-protected areas of the DAT-ON nucleus. (The ALLNUC parameter must be specified to obtain the entire nucleus, including the page-protected areas of the DAT-ON nucleus and the DAT-OFF nucleus.)

PSA The PSA for one processor is dumped. This is either the processor at the time of the error or the processor at the time of the dump.

RGN The allocated pages in the private area of each address space being dumped. This includes the following areas:

Subpools	Storage
0-127, 129-132, 229, 230, 240, 249, 250-252	All storage allocated to these subpools
203-205, 213-215, 223-225, 233-235, 253-255	All storage allocated to the LSQA and ELSQA
236, 237	All storage allocated to the SWA and ESWA

SVC dump does not dump all the obtained private storage in an address space when the RGN option of SDATA is specified. This reduces the number of page faults that occur during SVC dump processing, decreases the time required to take a dump, and reduces the size of the dump.

SVC dump dumps only those obtained pages that have something stored into them. Data-in-virtual pages that have been changed since the last DIV macro (that specified the SAVE service) executed, will also be dumped.

Eliminated from the dump are the pages of storage that are in a freshly-obtained state. Data-in-virtual pages that are in central storage, but have not been changed, are also considered to be in a freshly-obtained state. They will not be dumped.

SQA The SQA and ESQA subpools (226, 239, 245, 247, and 248).

SUMDUMP or SUM
A summary dump is to be included with the SVC dump output. The trace table is included in the nonsummary portion of the dump if this option is specified or used as a default.

The type of summary dump depends on how you specify the BRANCH and SUSPEND parameters:

- If you specify BRANCH=YES and SUSPEND=NO, you get a disabled summary dump.
- If you specify BRANCH=YES and SUSPEND=YES, you get a suspend summary dump.
- If you specify BRANCH=NO, you get an enabled summary dump.

For a description of the dump contents, see *OS/390 MVS Diagnosis: Tools and Service Aids*.

SWA The scheduler work area subpools for each address space being dumped (subpools 236 and 237). This includes all storage allocated above and below 16 megabytes.

TRT The system trace table, the GTF trace records, and master trace data if these types of traces are active

XESDATA Cross-system extended services (XES) information.

DEFAULTS or DEFS
The following default SDATA options are included in the SVC dump:
- ALLPSA
- SQA
- SUMDUMP
- IO
- Any default SDATA options specified by the CHNGDUMP command when CHNGDUMP is in ADD mode.

Notes:

1. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS is specified.
2. DEFAULTS and NODEFAULTS are mutually exclusive.

NODEFAULTS or NODEFS
The SDATA defaults are NOT included in the SVC dump. Specifying NODEFAULTS reduces the size of an SVC dump by excluding the following default SDATA options:
- ALLPSA
- SQA
- SUMDUMP
- IO
- Any default SDATA options specified by the CHNGDUMP command when CHNGDUMP is in ADD mode.

If a data area relating to an SDATA option is required in the dump, the programmer can code that SDATA option on the SDUMPX macro invocation. All keywords and SDATA options are valid when NODEFS is coded.

If you specify NODEFAULTS, the dump still contains some default system areas that are included in all dumps.

IO The IO data areas are included in the SVC dump.

,PROBDESC=*area addr*
Specifies the address of an area that contains information describing the problem. If register notation is used, the register contains the address of the area.

The area can be passed to any SVC dump, but is primarily intended for dumps requested by the REMOTE parameter. When a dump is requested through REMOTE, the system being dumped calls an IEASDUMP.QUERY routine. The routine uses the area to determine if its system should be dumped and, if so, what storage areas should be added to the dump.

The area is mapped by the IHASDPD mapping macro and must be in common storage. The area consists of:

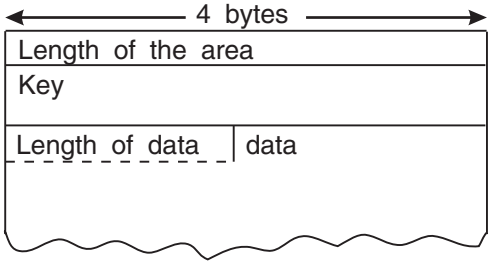
- **A 4-byte header**, which indicates the total length of the area. The length includes the 4 bytes of the header.
- **Key, length, data entries**. Each entry consists of:
  - Key: An 8-byte key, which you can use to identify the application or the problem or both. You might use the key field, for example, to contain a 3-byte identifier for the application and a 5-byte area for application information.

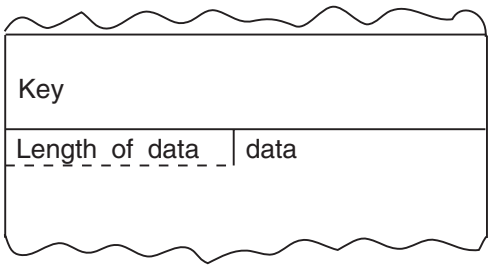
The key corresponds to the SDPD_KLD_KEY field in the IHASDPD mapping macro. The key must not begin with A through I or SYS; these are reserved for IBM use. IBM-supplied values for key are:

- SYSDCOND: Suppresses a dump on another system in a sysplex if the other system does not have an IEASDUMP.QUERY routine or if no IEASDUMP.QUERY routine returns a code of 0.
- SYSDLOCL: Requests a second, deferred dump of the local system, if the area for the REMOTE parameter specifies the local system. The deferred dump contains areas added by IEASDUMP.QUERY, IEASDUMP.GLOBAL, and IEASDUMP.LOCAL exit routines, if any routines had been associated with those exits.

– Length: A 2-byte field that gives the length of the data portion. The length does not include the length of this field itself or of the key field. A length of zero is valid.

– Data: The data portion, which consists of the number of bytes specified in the preceding field. You can place in the data portion any information you desire in any format.





If the area exceeds 1024 bytes, the caller must not free the area until data capture for the dump is completed. For an asynchronous dump, use an ECB or SRB parameter to be notified when data capture is complete or use a SUSPEND parameter to suspend other processing until data capture is completed. (For a synchronous dump on the local system, you do not have to worry about freeing the area too soon because the system does not return control to the caller until processing completes.) If the area exceeds 32,768 bytes, the system places in the dump only the first 32,768 bytes.

If the length is less than 4 bytes, the system ignores the PROBDESC parameter.

,STORAGE=*(strt addr,end addr)*
,LIST=*list addr*
,LISTA=*list addr*
,LISTD=*list addr*

Specifies one of the following:

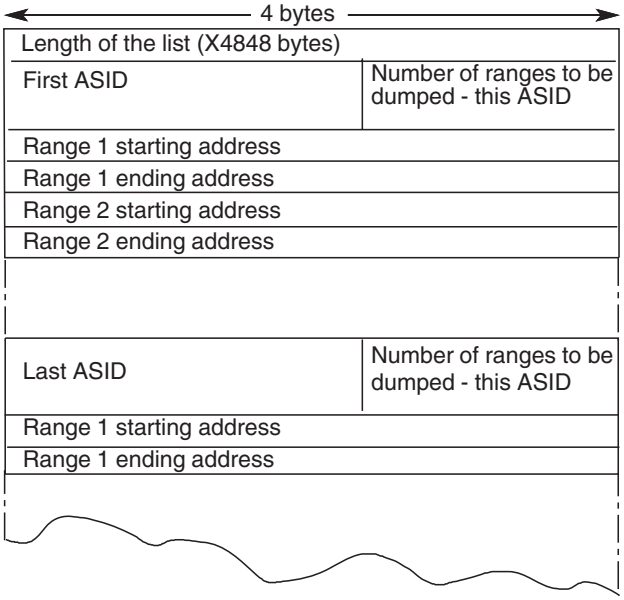
- one or more pairs of starting and ending addresses (STORAGE)

- a list of starting and ending addresses (LIST)
- a list of ASIDs and storage ranges (LISTA)
- a list of address ranges, qualified by STOKENs, of areas to be included in the SVC dump (LISTD).

Each starting address must be less than its corresponding ending address.

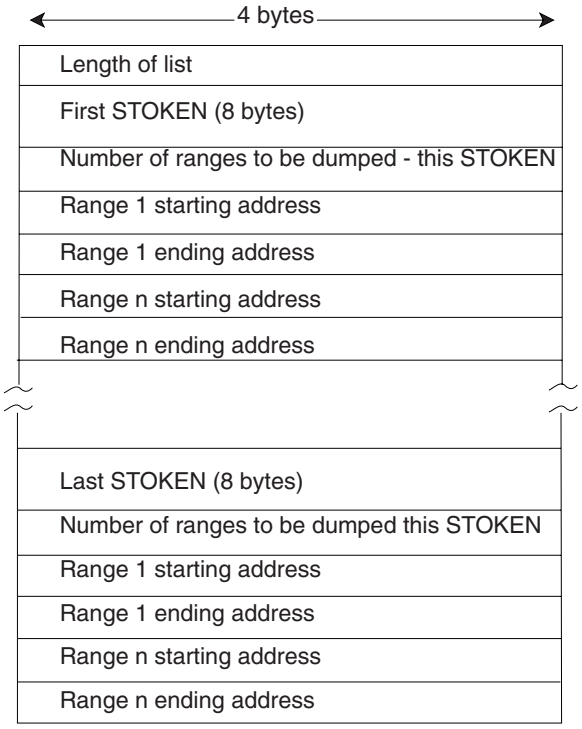
STORAGE and LIST: When STORAGE or LIST is specified, the list must contain an even number of addresses, and each address must occupy one fullword. In the list, the high-order bit of the fullword containing the last ending address of the list must be set to 1; all other high-order bits must be set to 0.

LISTA: When LISTA is specified, the first fullword of the storage list contains the number of bytes (including the first word) in the list. LISTA specifies a list of ASIDs and storage ranges as follows:



Note: If LISTA or SUBPLST is specified for a scheduled dump request and if the list does not exceed 484 bytes in size, SVC dump will move the list to SVC dump storage. The caller can free or reuse this space on return from SVC dump. No synchronization with SVC dump completion is required. If the list exceeds 484 bytes, SVC dump will not move the list and synchronization with SVC dump completion is required.

LISTD: When LISTD is specified, the first fullword of the list contains the number of bytes (including the first word) in the list. The list can be up to 5124 bytes (for a possible 256 single range entries). The number of entries processed will be fewer than 256, if either data spaces are requested using the DSPLIST parameter, or multiple ranges are specified for an STOKEN. For LISTD, specify the STOKENs and address ranges as follows:



STOKEN refers to any address/data space. SVC dump does not dump data space storage that has not been referenced.

LISTD causes a scheduled dump when the caller performs one of the following actions:

- Requests a SCOPE=SINGLE data space that is owned by a task in an address space other than the caller's primary address space.
- Requests an address space other than the primary

See the ASID parameter for the limit on address spaces that can be specified.

Dumps on Other Systems in a Sysplex: If the SDUMPX macro specifies SDRMT_IDCON_STORAGE_COPY in the area specified by the REMOTE parameter, the dumps requested by REMOTE contain areas specified by LISTD on this macro; the dumps do not contain areas specified by LIST and LISTA. If an STOKEN is all zeros in the area specified by the REMOTE parameter, the dumps include the indicated address range within all address spaces included in the dump.

If the LISTD area exceeds 484 bytes, only the data requested in the first 484 bytes is included in the dumps requested by REMOTE.

,SUBPLST=*subpool id list address*

Specifies a list of ASIDs with associated subpool ids corresponding to subpools of virtual storage that are to be included in the SVC dump.

The first fullword of the list contains the number of bytes (including the first word) in the list. The list can contain a maximum of 200 bytes consisting of unique ASIDs and subpool IDs. If the list contains duplicate ASIDs or subpool IDs, the length can exceed 200 bytes because SDUMPX stores the unique subpool IDs in a 200-byte work area.

The structure of the list for each ASID follows:

- The first word contains the ASID in bits 0-15 and the number of subpools associated with this ASID (n) in bits 16-31. If 0 is specified as the ASID, the caller's home ASID is used.
- The next n halfwords contain the subpool IDs (right justified) corresponding to the virtual storage to be included in the SVC dump. The manner in which these

subpools are dumped depends on whether they are private or common area subpools.

- If a private area subpool (related to a TCB) is specified, all virtual storage associated with this subpool, for all TCBs in the specified address space, is dumped.
- If a common area subpool is specified, all of the virtual storage allocated in the subpool is dumped.

SVC dump does not dump all the obtained storage in an address space if the SUBPLST list keyword for private subpools is coded. This reduces the number of page faults that occur during SVC dump processing and the time required to take a dump. It also reduces the size of dumps on tape or DASD.

For storage that is not related to data-in-virtual, only obtained pages that have something stored into them are dumped. This eliminates the pages of storage that are in a freshly obtained state.

For storage that is related to data-in-virtual, only pages that are in central storage are dumped, as well as pages that have been changed since the last data-in-virtual SAVE operation.

Notes:

1. SVC dump ignores unassigned subpool IDs and ASIDs.
2. If an invalid subpool or ASID (ASID greater than ASVTMAXU) is specified, the caller receives a 233 ABEND and SDUMP processing terminates the dump.
3. If all ASIDs specified in SUBPLST are the current ASID, SUBPLST does not force a scheduled dump. However, if any of the ASIDs are different, a scheduled (or asynchronous) dump results.
4. SDUMPX callers executing in key 0 and supervisor state, who request storage from subpool 0 via GETMAIN obtain that storage from subpool 252 instead. Therefore, when these callers want to dump this storage, they must specify subpool 252 rather than subpool 0.

,KEYLIST=*storage key list addr*

Specifies the address of a list of storage keys associated with the virtual storage to be dumped. If the key of a subpool specified in SUBPLST does not match a key in this list, the data in the subpool is not dumped. SUBPLST must be specified if the KEYLIST option is used. If you do not specify KEYLIST, all storage (regardless of key) associated with the requested subpools is included in the dump. Therefore, if you want to dump the storage corresponding to all 16 keys, do not specify this parameter.

The list contains one-byte entries and starts on a halfword boundary. The first byte indicates the length of the list (including this byte). The list has a maximum length of 16 bytes so that up to 15 keys can be specified. Specify each key in the left-most four bits of each byte, except the length byte.

,BUFFER=NO
,BUFFER=YES

Specifies that the contents of the SQA buffer is (YES) or is not (NO) to be included in the dump. (The SQA buffer does not include the SDUMPX parameter list or any data pointed to by the parameter list.) Callers who specify BUFFER=YES on the SDUMPX macro will obtain a dump of a 4K buffer reserved in the SQA for the callers of SVC dump. You can reserve the buffer by setting the high-order bit of the CVTSDBF field in the communications vector table (CVT). Once you have reserved the buffer, you can fill it with data before issuing SDUMPX. Programs that are involved with data that might change before SDUMPX can dump it should use this buffer.

The CVTSDBF field of the CVT points to the buffer. Before using the buffer, use compare and swap logic to serialize on the high-order bit of CVTSDBF. If the bit was on (B'1'), the buffer is in use, and you should continue processing as though a dump could not be taken. If the bit was off (B'0'), the bit is set to B'1' by the compare and swap. You

can then fill the buffer and issue SDUMPX. If the compare and swap sets the CVTSDBF bit, SDUMPX resets it for you.

,QUIESCE=YES
,QUIESCE=NO
Specifies that the system is to be set nondispatchable until the contents of the SQA and the CSA are dumped (YES), or that the system is to be left dispatchable (NO). If the SDATA parameter does not specify SQA or CSA, the QUIESCE=YES request is ignored.

Note: Summary dumps (SUMDUMP) for branch entries (BRANCH=YES) always cause the system to be set nondispatchable until the summary dump is written.

,BRANCH=NO
,BRANCH=YES
Specifies that a branch entry is to be used for interfacing with SVC dump to schedule a dump (YES), or that an SVC instruction is to be generated for interfacing with SVC dump (NO).

For BRANCH=YES, MVS produces a scheduled (asynchronous) SVC dump. For BRANCH=NO, the parameters you code to identify storage determine whether MVS produces a scheduled or synchronous SVC dump. MVS produces a scheduled dump when you code BRANCH=NO with one or more of the following:

- ASIDLST
- ASID=*asid addr*
- TYPE=XMEM or TYPE=XMEME
- LISTA
- LISTD=*list addr*, when the STOKEN represents either an address space other than the primary address space, or a SCOPE=SINGLE data space owned by a program that is not running in the primary address space
- SUBPLST=*subpool id list addr*, when the list of address spaces with associated subpool IDs contains at least one address space other than the primary address space.

You might make different design decisions for your program based on the type of dump MVS produces. See *OS/390 MVS Programming: Authorized Assembler Services Guide* for guidance about designing your program to handle each type of dump.

If BRANCH=YES is specified and the caller has not specified at least one of the following keywords: ASID, ASIDLST, TYPE=XMEM, TYPE=XMEME, or LISTA, the dump is scheduled to the home address space.

Routines that issue SDUMPX with BRANCH=YES must provide a 72-byte save area pointed to by register 13, and must include the CVT mapping macro.

For BRANCH=YES entry by reentrant recovery routines, SDUMPX processing moves the data supplied by the following parameters to a system area:

HDR
HDRAD
ID
IDAD
ASIDLST
STORAGE
LIST
LISTA
SUBPLST
KEYLIST

This enables the recovery routine to free its storage on return from SDUMPX although the dump has not completed.

,SUSPEND=NO
,SUSPEND=YES
Specifies that a suspend summary dump is requested (YES) or not requested (NO). SUSPEND=YES must be used together with the BRANCH=YES and SDATA=SUMDUMP parameters. This keyword should be used by routines that can experience page faults but that want to save dump information in a virtual storage buffer.

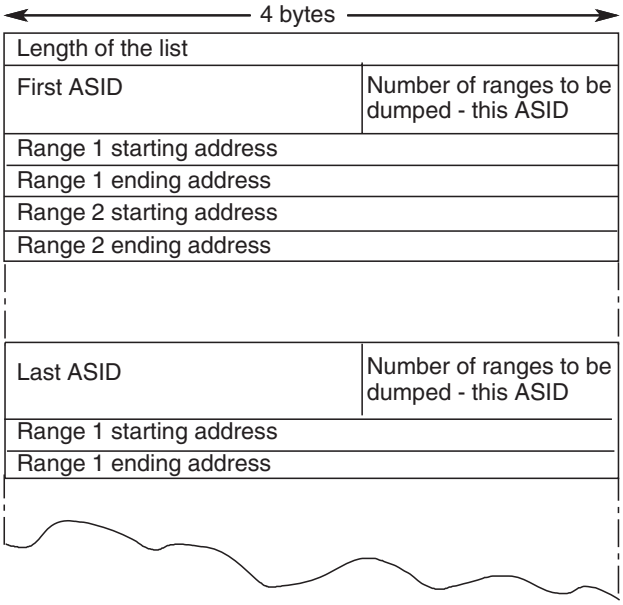
,SUMLIST=*list addr*
,SUMLSTA=*list addr*
,SUMLSTL=*list addr*
Specifies one of the following:

- a list of starting and ending addresses of areas to be included in a summary dump (SUMLIST)
- a combined list of ASIDs and storage ranges (SUMLSTA)
- a list of address ranges, qualified by ALETs, of areas to be included in a summary dump.

SUMDUMP must be specified as an SDATA parameter and each starting address must be less than its corresponding ending address.

SUMLIST: For SUMLIST, the storage list must contain an even number of addresses, and each address must occupy one fullword. In the list, the high-order bit of the fullword containing the last ending address of the list must be set to 1, and all other high-order bits must be set to 0.

SUMLSTA: For SUMLSTA, the first fullword of the list contains the number of bytes (including the first word) in the list. SUMLSTA specifies a list of ASIDs and storage ranges as follows:



Restriction:

- The maximum number of ASIDs that the combined TYPE=XMEM, TYPE=XMEME, LISTA, ASIDLST, ASID, and SUBPLST parameters can specify is fifteen.

  Note: There is no restriction on the number of ASIDs that the SUMLSTA can specify.

When BRANCH=YES and SUSPEND=NO are also specified, the list must be addressable using the addressability established when SVC dump was entered. The

lists themselves and all ranges specified must reference paged-in data. Paged-out data is not dumped by summary dump.

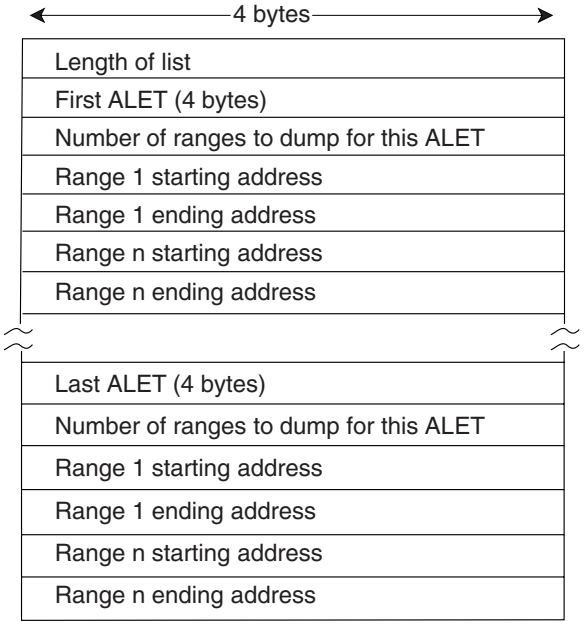
When BRANCH=YES and SUSPEND=YES are also specified, the lists must be addressable using the addressability established when SVC dump was entered. The lists and referenced data can either be in paged in or paged out areas. The maximum amount of summary dump data with this type of dump is 1M.

When BRANCH=NO is also specified, the lists must be addressable in all address spaces in which the dump will be taken (those address spaces specified by ASID, ASIDLST, LISTA, or TYPE=XMEM, TYPE=XMEME, or SUBPLST). The lists and referenced data can be in paged-in or paged-out areas. The maximum amount of summary dump data possible with this type of dump is dependent only on the size of the dump data set.

Each ASID specified with SUMLSTA must represent a valid, swapped-in address space in order for the data to be dumped.

Programming Notes: The total number of distinct ASIDs that can be specified by TYPE=XMEM, TYPE=XMEME, LISTA, ASID, SUBPLST and ASIDLST is fifteen. If more than fifteen are requested, only the first fifteen are processed. There is no restriction on the number of ASIDs specified by the SUMLSTA parameter, nor do SUMLSTA ASIDs contribute toward the fifteen ASID limit.

SUMLSTL: For SUMLSTL, the first fullword of the list contains the number of bytes (including the first word) in the list. Specify the ALETs and address ranges as follows:



ALET refers to entries in either a DU-AL or a PASN-AL, and associated with any address/data space that the caller has addressability to. SVC dump does not dump data space storage that has not been referenced.

Return and Reason Codes

The following tables identify return codes and reason codes, tell what each means, and recommend actions that you should take.

Register 15 Return Codes

If BRANCH=NO was specified and no ASIDs other than the current ASID were requested, register 15 contains one of the following hexadecimal return codes when control is returned at the completion of the capture phase:

Figure 58. Return Codes for the SDUMPX Macro when BRANCH=NO

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: A complete dump was taken. **Action**: None
04	**Meaning**: A partial dump was taken because the dump data set did not have sufficient space. **Action**: Examine the reason code that explains why a partial dump was taken. The reason code is contained in message IEA911E. If you specified the SRB parameter, you can include the IHASDRSN mapping macro to map the reason code information.
08	**Meaning**: The system was unable to take a dump. **Action**: Examine the reason code that explains why no dump was taken (see "Reason Codes for Return Code 08" on page 299).

If BRANCH=YES or any ASID other than the current ASID was requested, register 15 contains one of the following hexadecimal return codes when control is returned after the system has scheduled the dump:

Figure 59. Return Codes for the SDUMPX Macro when BRANCH=YES

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: A dump was scheduled. **Action**: None
08	**Meaning**: The system was unable to schedule a dump. **Action**: Examine the reason code that explains why no dump was taken (see "Reason Codes for Return Code 08" on page 299).

ECB and SRB Return Codes

If ECB=(*ecb addr*), ECB=(*ecb addr*,CAPTURE), SRB=(*srb addr*), or SRB=(*srb addr*,CAPTURE), was specified and the DCB parameter was not specified, the system also returns one of following hexadecimal codes in the ECB or SRB at the completion of the capture phase:

Figure 60. Return Codes for the ECB Parameter and SRB Parameter

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: All the requested data was captured and the dump writing phase was successfully initiated. **Action**: None
04	**Meaning**: Some of the requested data could not be captured and one or more partial dump indicators have been set in SDRSN. The dump writing phase was successfully initiated. **Action**: Examine the reason code that explains why a partial dump was taken. The reason code is contained in message IEA911E. If you specified the SRB parameter, you can include the IHASDRSN mapping macro to map the reason code information.
08	**Meaning**: The system was unable to take a dump. **Action**: Examine the reason code that explains why no dump was taken (see "Reason Codes for Return Code 08" on page 299).

If ECB=(*ecb addr*,WRITE) or SRB=(*srb addr*,WRITE) was specified, or if any of the options on the ECB or SRB parameters were specified along with the DCB parameter, the system also returns one of the following hexadecimal codes in the ECB or SRB at the completion of the dump writing phase:

Figure 61. Return Codes for the ECB or SRB Parameter with the DCB Parameter	
Hexadecimal Return Code	**Meaning and Action**
00	**Meaning**: All the requested data was captured and then written to the dump data set. **Action**: None
04	**Meaning**: Some of the requested data could not be captured or could not be written to the dump data set. **Action**: Examine the reason code that explains why a partial dump was taken. The reason code is contained in message IEA911E. If you specified the SRB parameter, you can include the IHASDRSN mapping macro to map the reason code information. The reason codes might also be passed to the SRB routine in the SDSTPDRC field of SDSTATUS.
08	**Meaning**: The system was unable to take a dump. **Action**: Examine the reason code that explains why no dump was taken (see "Reason Codes for Return Code 08" on page 299).

Note: The ECB will not be posted unless the return code from SDUMPX is 0.

Reason Codes for Return Code 08

When a return code of 08 is received, a hexadecimal reason code is returned. The reason code is in the following locations:

- In the SDWASDRC field of the SDWA if you issued SDUMPX in a recovery routine, and the system provided an SDWA.
- In the ECB or register 15 (bits 16-23), provided that the FAILRC parameter is specified.
- In the SDSTATUS field. This field is pointed to by the SRBPARM field that is in the SRB parameter list. The parameter list is passed to SDUMPX by using the SRB keyword.

The reason codes are as follows:

Figure 62 (Page 1 of 4). Reason Codes for Return Code 08	
Hexadecimal Reason Code	**Meaning and Action**
0	**Meaning**: No SVC dump was requested. **Action**: None
1	**Meaning**: An SVC dump was successfully started. **Action**: None
2	**Meaning**: An SVC dump was suppressed because the capture phase of another SVC dump was in progress. **Action**: Wait until the dump in progress has been captured (as identified by message IEA794I) and reissue SDUMPX.
3	**Meaning**: An SVC dump was suppressed by a request by the installation (for example: DUMP=NO at IPL or CHNGDUMP SET,NODUMP). **Action**: Issue CHNGDUMP SET,SDUMP or CHNGDUMP RESET,SDUMP and reissue SDUMPX.
4	**Meaning**: An SVC dump was suppressed by a SLIP NODUMP command. **Action**: Delete SLIP trap with SLIP DEL command and reissue SDUMPX.
5	**Meaning**: An SVC dump was suppressed because a SYS1.DUMP data set was not available. **Action**: If MSGTIME expired, increase MSGTIME limit with CD SET,SDUMP,MSGTIME= command. Make a dump dataset available via the DUMPDS ADD,DSN= and/or DUMPDS CLEAR,DSN= commands and reissue SDUMPX.
6	**Meaning**: An SVC dump was suppressed because an I/O error occurred during the initialization of the SYS1.DUMP data set. **Action**: Reissue SDUMPX.
8	**Meaning**: An SVC dump was suppressed because an SRB could not be scheduled to activate the dump tasks in the requested address spaces. **Action**: None

Figure 62 (Page 2 of 4). Reason Codes for Return Code 08

Hexadecimal Reason Code	Meaning and Action
9	**Meaning**: An SVC dump was suppressed because a terminating error occurred in SVC dump before the first dump record was written. **Action**: Reissue SDUMPX.
A	**Meaning**: An SVC dump was suppressed because a status stop SRB condition was detected. **Action**: None
B	**Meaning**: An SVC dump was suppressed by DAE. **Action**: None.
15	**Meaning**: The parameter list address is zero. **Action**: Supply a parameter list address in register 1 and reissue SDUMPX.
16	**Meaning**: The parameter list is not a valid SVC or SNAP parameter list. **Action**: Provide the address of a valid SVC dump parameter list in register 1 and reissue SDUMPX.
17	**Meaning**: The caller-supplied data set is not supported. **Action**: Supply a dataset with LRECL >= 4160 open with EXCP on a device supported by SVC dump (or use a system dump dataset) and reissue SDUMPX.
18	**Meaning**: The start address is greater than or equal to the end address in a storage list. **Action**: Correct the address range that is not valid and reissue SDUMPX.
19	**Meaning**: The caller-supplied header is longer than 100 characters. **Action**: Supply a shorter header and reissue SDUMPX.
1A	**Meaning**: The caller requested a 4K buffer, but did not reserve it. **Action**: Set the high-order bit of CVTSDBF and reissue SDUMPX.
1B	**Meaning**: A storage list overlaps the 4K buffer. **Action**: Move the storage list so that it does not overlap the SVC dump 4K buffer pointed to by CVTSDBF. Reissue SDUMPX.
1C	**Meaning**: The caller-supplied DCB is not valid. **Action**: Make sure DCB is open, does not overlap 4K buffer, and represents a tape or DASD dataset, then reissue SDUMPX.
1E	**Meaning**: An ASID in the ASID list is syntactically not valid. **Action**: Supply a valid ASID (<= ASVTMAXU) and reissue SDUMPX.
22	**Meaning**: The 4K buffer was requested with an SVC dump already in progress. **Action**: Wait until the dump in progress has been captured and reissue SDUMPX.
25	**Meaning**: A subpool ID that was not valid was specified in the subpool list. **Action**: Supply a valid subpool id (<= 255) and reissue SDUMPX.
28	**Meaning**: Part of the parameter list is inaccessible. **Action**: Make sure the parameter list is addressable from the caller's current address space. Reissue SDUMPX.
29	**Meaning**: The caller-supplied DCB is inaccessible. **Action**: Make sure the DCB is addressable from the caller's current address space. Reissue SDUMPX.
2A	**Meaning**: The caller-supplied storage list is inaccessible. **Action**: Make sure the storage list addressable from the caller's current address space. Reissue SDUMPX.
2B	**Meaning**: The caller-supplied header data is inaccessible. **Action**: Make sure the header is addressable from the caller's current address space. Reissue SDUMPX.
2C	**Meaning**: The caller-supplied ECB is inaccessible. **Action**: Make sure the ECB is addressable from the caller's current address space. Reissue SDUMPX.
2D	**Meaning**: The caller's ASID list is inaccessible. **Action**: Make sure the ASID list is addressable from the caller's current address space. Reissue SDUMPX.

Figure 62 (Page 3 of 4). Reason Codes for Return Code 08

Hexadecimal Reason Code	Meaning and Action
2E	**Meaning**: The caller's SUMLIST/SUMLSTA is inaccessible. **Action**: Make sure the SUMLIST/SUMLSTA is addressable from the caller's current address space. Reissue SDUMPX.
2F	**Meaning**: The caller's SUBPLST list is inaccessible. **Action**: Make sure the SUBPLST is addressable from the caller's current address space. Reissue SDUMPX.
30	**Meaning**: The caller's KEYLIST is inaccessible. **Action**: Make sure the KEYLIST is addressable from the caller's current address space. Reissue SDUMPX.
31	**Meaning**: Copies of the SLIP register and PSW are inaccessible. **Action**: None
32	**Meaning**: The caller-supplied SRB is inaccessible. **Action**: Make sure the SRB is addressable from the caller's current address space. Reissue SDUMPX.
33	**Meaning**: The version number in the parameter list is not valid. **Action**: Supply a parameter list with a valid version number and reissue SDUMPX.
34	**Meaning**: The caller's LISTD is inaccessible. **Action**: Make sure the LISTD is addressable from the caller's current address space. Reissue SDUMPX.
35	**Meaning**: The caller's SUMLISTL is inaccessible. **Action**: Make sure the SUMLISTL is addressable from the caller's current address space. Reissue SDUMPX.
36	**Meaning**: The parameter list contains conflicting parameters. **Action**: Remove the conflicting parameters (for example, both ECB and SRB specified) and reissue SDUMPX.
37	**Meaning**: The ID is longer than 50 characters. **Action**: Supply a shorter ID and reissue SDUMPX.
38	**Meaning**: The ID is not addressable. **Action**: Make sure the ID is addressable from the caller's current address space. Reissue SDUMPX.
39	**Meaning**: The PSWREGS area is an incorrect length. **Action**: Correct the length of the PSWREGS area and reissue SDUMPX.
3A	**Meaning**: The PSWREGS area is not addressable. **Action**: Make sure the PSWREGS area is addressable from the caller's current address space. Reissue SDUMPX.
3B	**Meaning**: The symptom record is not valid. **Action**: Supply a valid symptom record and reissue SDUMPX.
3C	**Meaning**: The symptom record is not addressable. **Action**: Make sure the symptom record is addressable from the caller's current address space. Reissue SDUMPX.
3D	**Meaning**: The DEB for the caller-supplied DCB is inaccessible. **Action**: Make sure the DEB for the caller-supplied DCB is addressable from the caller's current address space. Reissue SDUMPX.
3E	**Meaning**: SVC dump is already using the maximum amount of virtual storage (as determined by the installation, using the MAXSPACE parameter on the CHNGDUMP command) to process other dumps. **Action**: Make a dump dataset available via the DUMPDS ADD,DSN= or DUMPDS CLEAR,DSN= command, reply DELETE to an outstanding IEA793A message, or increase the amount of virtual storage that SDUMPX is allowed to use via the CHNGDUMP SET,SDUMP,MAXSPACE= command, then reissue SDUMPX.
3F	**Meaning**: The caller-supplied STRLIST area is inaccessible. **Action**: Make sure the STRLIST area is addressable from the caller's current address space. Reissue SDUMPX.

Figure 62 (Page 4 of 4). Reason Codes for Return Code 08

Hexadecimal Reason Code	Meaning and Action
40	**Meaning**: The caller-supplied INTOKEN area is inaccessible. **Action**: Make sure the INTOKEN area is addressable from the caller's current address space. Reissue SDUMPX.
41	**Meaning**: The caller-supplied REMOTE area is inaccessible. **Action**: Make sure the REMOTE area is addressable from the caller's current address space. Reissue SDUMPX.
42	**Meaning**: The caller-supplied PROBDESC area is inaccessible. **Action**: Make sure the PROBDESC area is addressable from the caller's current address space. Reissue SDUMPX.
43	**Meaning**: The caller-supplied JOBLIST area is inaccessible. **Action**: Make sure the JOBLIST area is addressable from the caller's current address space. Reissue SDUMPX.
44	**Meaning**: The caller-supplied DSPLIST area is inaccessible. **Action**: Make sure the DSPLIST area is addressable from the caller's current address space. Reissue SDUMPX.
45	**Meaning**: The caller-supplied REMOTE area is not valid. The length of a field in the REMOTE area is specified as less than 4 bytes. Note that, if the length of the entire REMOTE area is less than 4 bytes, the REMOTE parameter is ignored. **Action**: Correct the lengths specified in the area mapped by the IEASDRMT macro. Reissue SDUMPX.
46	**Meaning**: SVC dump stopped the dump because the system resources manager (SRM) detected a critical shortage of auxiliary storage. **Action**: See the system programmer response for message IRA201E to determine how to relieve the shortage. Then reissue the SDUMPX macro.

SDUMPX—List Form

Use the list form of the SDUMPX 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 SDUMPX must be equal to the maximum number of addresses specified in the execute form of the macro.

Syntax

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

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: From 1 to 100 characters. *dump title addr*: A-type address.
,DCB=*dcb addr*	*dcb addr*: A-type address.
,JOBLIST=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,DSPLIST=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,PLISTVER=1 ,PLISTVER=2	*decimal digit 1*: Use up to 112-byte parameter list. *decimal digit 2*: Use 128-byte parameter list.

,PLISTVER=3	*decimal digit 3*: Use 184-byte parameter list. **Default**: PLISTVER=2, if you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFS, SDATA=NODEFS, or SDATA=IO. PLISTVER=3, if you specify STRLIST, REMOTE, INTOKEN, DSPLIST, JOBLIST, or PROBDESC. Otherwise, the default is PLISTVER=1.
,SYMREC=*symrec addr*	*symrec addr*: RX-type address, or register (2) - (12).
,ID='*identifier*' ,IDAD=*identifier addr*	*identifier*: From 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,INTOKEN=*token addr*	*token addr*: RX-type address, or register (2) - (12).
,REMOTE=*area addr*	*area addr*: RX-type address, or register (2) - (12).
,STRLIST-*structure list addr*	*structure list addr*: RX-type address or register (2) - (12).
,PSWREGS=*parm list addr*	*parm list addr*: RX-type address, or register (2) - (12).
,SDATA=*(sdata options)*	*sdata options*: Any combination of the following, separated by commas: ALLNUC, ALLPSA, COUPLE, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT, XESDATA, DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO **Notes:** 1. Executing the SDUMPX macro causes ALLPSA, SQA, IO, and SUMDUMP storage areas to be dumped unless excluded by NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP. 2. The PSA and IO options are not required unless NODEFAULTS is specified, because they are dumped as a default in all SVC dumps. 3. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS has been specified.
,PROBDESC=*area addr*	*area addr*: RX-type address, or register (2) - (12).
,STORAGE=*(strt addr,end addr)*	*strt addr*: A-type address. *end addr*: A-type address.
,LIST=*list addr* ,LISTA=*list addr* ,LISTD=*list addr*	*list addr*: A-type address. **Note:** Specify one or more pairs of addresses, separated by commas.
,SUBPLST=*subpool id list addr*	*subpool id list addr*: A-type address, or register (2) - (12).
,KEYLIST=*storage key list addr*	*storage key list addr*: A-type address, or register (2) - (12). **Note**: KEYLIST cannot be specified without SUBPLST.
,BUFFER=NO ,BUFFER=YES	**Default**: BUFFER=NO
,QUIESCE=YES ,QUIESCE=NO	**Default**: QUIESCE=YES
,TYPE=*(type code)*	*type code*: Any combination of the following, separated by commas: XMEM or XMEME, NOLOCAL.
,MF=L	

Parameters

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

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

Note: If SYMREC, ID, IDAD, PSWREGS, SDATA=NODEFS, SDATA=DEFS or SDATA=IO is not used on the list form of the macro, but is coded on the execute form, use PLISTVER=2 when specifying MF=L to generate a 128-byte parameter list.

SDUMPX—Execute Form

A remote control program parameter list is referred to and can be modified by the execute form of the SDUMPX macro.

If you code one or more of the SDATA parameters on the execute form of the macro, any SDATA parameters coded on the list form are lost.

Syntax

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

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

HDR=*'dump title'* HDRAD=*dump title addr*	*dump title*: From 1 to 100 characters. *dump title addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,ASID=*ASID addr* ,ASIDLST=*list addr*	*ASID addr*: A-type address, or register (2) - (12). *list addr*: RX-type address, or register (2) - (12).
,JOBLIST=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,DSPLIST=*list addr*	*list addr*: RX-type address, or register (2) - (12).
,TYPE=*(type code)*	*type code*: Any of the following, separated by commas: XMEM, XMEME, NOLOCAL, FAILRC **Note**: XMEM and XMEME are mutually exclusive codes.
,PLISTVER=1 ,PLISTVER=2 ,PLISTVER=3	*decimal digit 1*: Use up to 112-byte parameter list. *decimal digit 2*: Use 128-byte parameter list. *decimal digit 3*: Use 184-byte parameter list. **Default**: PLISTVER=2, if you specify SYMREC, ID, IDAD, PSWREGS, SDATA=DEFS, SDATA=NODEFS, or SDATA=IO. PLISTVER=3, if you specify STRLIST, REMOTE, INTOKEN, DSPLIST, JOBLIST, or PROBDESC. Otherwise, the default is PLISTVER=1.
,SYMREC=*symrec addr*	*symrec addr*: RX-type address, or register (2) - (12).
,ID='*identifier*' ,IDAD=*identifier addr*	*identifier*: From 1 to 50 characters. *identifier addr*: RX-type address, or register (2) - (12).
,INTOKEN=*token addr*	*token addr*: RX-type address, or register (2) - (12).

,REMOTE=*area addr*

area addr: RX-type address, or register (2) - (12).

,STRLIST-*structure list addr*

structure list addr: RX-type address or register (2) - (12).

,PSWREGS=*parm list addr*

parm list addr: RX-type address, or register (2) - (12).

,ECB=(*ecb addr*)
,ECB=(*ecb addr*,CAPTURE)
,ECB=(*ecb addr*,WRITE)
,SRB=(*srb addr*)
,SRB=(*srb addr*,CAPTURE)

ecb addr: A-type address, or register (2) - (12).

srb addr: A-type address, or register (2) - (12).
Notes:

1. If you code ECB=(*ecb addr*), without specifying CAPTURE or WRITE, SVC dump posts the ECB at the completion of the capture phase unless you also specify the DCB parameter. If you specify both the ECB and DCB parameters, the ECB is posted at the completion of the writing phase.
2. If you code SRB=(*srb addr*), without specifying CAPTURE or WRITE, SVC dump schedules the SRB at the completion of the capture phase unless you also specify the DCB parameter. If you specify both the SRB and DCB parameters, the SRB is scheduled at the completion of the writing phase.

,SRB=(*ecb addr*,WRITE)

,SDATA=*(sdata options)*

sdata options: Any combination of the following, separated by commas:
ALLNUC, ALLPSA, COUPLE, CSA, GRSQ, LPA, LSQA, NOALLPSA/NOALL, NOSQA, NOSUMDUMP/NOSUM, NUC, PSA, RGN, SQA, SUMDUMP/SUM, SWA, TRT, XESDATA, DEFAULTS/DEFS, NODEFAULTS/NODEFS, IO
Notes:

1. Executing SDUMPX causes ALLPSA, SQA, IO, and SUMDUMP storage areas to be dumped unless excluded by the NOALLPSA, NOSQA, NODEFAULTS, or NOSUMDUMP parameter.
2. The PSA and IO options are not required unless NODEFS is specified, because they are dumped as a default in all SVC dumps.
3. DEFAULTS is not required. All SVC dumps include the default SDATA options unless NODEFAULTS has been specified.

,PROBDESC=*area addr*

area addr: RX-type address, or register (2) - (12).

,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*
,LISTA=*list addr*

list addr: A-type address, or register (2) - (12).
Note: Specify one or more pairs of addresses, separated by commas.

,LISTD=*list addr*

,SUBPLST=*subpool id list addr*

subpool id list addr: RX-type address, or register (2) - (12).

,KEYLIST=*storage key list addr*

storage key list addr: RX-type address, or register (2) - (12).

Note: KEYLIST cannot be specified without SUBPLST.

,BUFFER=NO
,BUFFER=YES

Default: BUFFER=NO

,QUIESCE=YES
,QUIESCE=NO

Default: QUIESCE=YES

,BRANCH=NO

Default: BRANCH=NO

,BRANCH=YES	**Note**: If BRANCH=YES is specified, ASID or ASIDLST must also be specified.
,SUSPEND=NO ,SUSPEND=YES	**Default**: SUSPEND=NO
,SUMLIST=*list addr* ,SUMLSTA=*list addr* ,SUMLSTL=*list addr*	*list addr*: RX-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 SDUMPX macro, with the following exception:

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

Appendix A. 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.

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:

© Copyright IBM Corp. 1988, 2000

IBM Corporation
Mail Station P300
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming Interface Information

This book is intended to help the customer to code macros that are available to authorized assembler language programs. This book documents intended Programming Interfaces that allow the customer to write programs to obtain services of OS/390.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

- IBM
- IBMLink
- MVS/ESA
- MVS/SP
- MVS/XA
- OS/390
- RACF
- SP
- SP1
- SP2

Other company, product and service names may be the trademarks or service marks of others.

Index

A

addressing mode and the services 1
ALET qualification
of parameters 3
AR (access register) mode
description 2
ASC (address space control) mode
defining 2
ASCB (address space control block)
locating 45
asynchronous execution
scheduling system services 246

C

callable service
coding 16
coding the callable services 16
coding the macros 13
command input buffer
manipulating 201
continuation line 15
CVT (communications vector table)
CVTSDBF field 264, 294

D

downward incompatible macro
list 7
dynamic output
text unit
pointer list 124

E

event
signalling completion 183

F

fast path page service 179
FRACHECK macro 203

I

initialize asynchronous exits 239
internal START command 81
issue
remote immediate signal 235

L

linkage index
freeing 49
reserving 53
LLACOPY macro 25
LOAD macro 31
load module
bringing into virtual storage 31
LOADWAIT macro 39
LOCASCB macro 45
lock
providing
via an NI instruction 105
via an OI instruction 117
LXFRE macro 49
LXRES macro 53

M

macro
coding 13
forms 12
level
selecting 7
sample 14
selecting level 7
user parameter, passing 3
X-macros
using 11
MCSOPER macro 57
MCSOPMSG macro 73
MGCR macro 81
MGCRE macro 85
MIHQUERY macro 91
MODESET macro 97

N

NIL macro 105
NML (nucleus module list) 109
NMLDEF macro 109
Notices 307
nucleus map lookup service 113
nucleus module list
See also NMLDEF macro
description 109
NUCLKUP macro 113

O

OIL macro 117

© Copyright IBM Corp. 1988, 2000

OUTADD macro 121
OUTDEL macro 135
output descriptor
creating 121
deleting 135
system-generated name 123

P

page service 171
parameter list
length for SDUMP macro 254
length for SDUMPX macro 279
PCLINK macro 147
PGANY macro 157
PGFIX macro
contents
fixing 159
PGFIXA macro 163
PGFREE macro 165
PGFREEA macro 169
PGSER macro 171, 179
PGSER macro (fast path) 179
POST macro 183
program call
linkage information
EXTRACT 152
STACK 147
UNSTACK 149
PTRACE macro 191
PURGEDQ macro 195

Q

QEDIT macro 201

R

RACDEF macro 203
RACF macros 203
RACHECK macro 203
RACINIT macro 203
RACLIST macro 203
RACROUTE macro 203
RACSTAT macro 203
RACXTRT macro 203
RESERVE macro 205
RESMGR macro 217
RESUME macro for RBs 227
RESUME macro for SRBs 231
RISGNL macro 235

S

SCHEDIRB macro 239
schedule asynchronous exits 239
SCHEDULE macro 245
schedule system services for asynchronous execution 246
SCHEDXIT macro 251
SDUMP macro 253
calculating parameter list length 254
in a reentrant program 254
SDUMPX macro 277
calculating parameter list length 279
in a reentrant program 279
service
ALET qualification 3
summary 17
services
addressing mode 1
ASC mode
defining 2
using 1
shared DASD
reserve a device 205
SQA buffer
dumped by SDUMPX 264, 294
SRB (service request block)
purging activity 195
START command
internal 81
SYS1.NUCLEUS parmlib member
with NMLs 109
system status
changing 97

U

user parameter
passing 3

V

virtual storage
bringing in a load module 31
contents
fix 163
free 165
dumping 253, 277
freeing contents 169

W

wait state
putting the system into 39

X

X-macros
using 11

Communicating Your Comments to IBM

OS/390
MVS Programming: Authorized
Assembler Services Reference, Volume 3
(LLACOPY-SDUMPX)

Publication No. GC28-1766-08

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM. Whichever method you choose, make sure you send your name, address, and telephone number if you would like a reply.

Feel free to comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. However, the comments you send should pertain to only the information in this manual and the way in which the information is presented. To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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.

If you are mailing a reader's comment form (RCF) from a country other than the United States, you can give the RCF to the local IBM branch office or IBM representative for postage-paid mailing.

- If you prefer to send comments by mail, use the RCF at the back of this book.
- If you prefer to send comments by FAX, use this number:
  - FAX: (International Access Code)+1+845+432-9405
- If you prefer to send comments electronically, use one of these network IDs:
  - Internet e-mail: mhvrcfs@us.ibm.com
  - World Wide Web: http://www.ibm.com/s390/os390/webqs.html

Make sure to include the following in your note:

- Title and publication number of this book
- Page number or topic to which your comment applies

Optionally, if you include your telephone number, we will be able to respond to your comments by phone.

Reader's Comments — We'd Like to Hear from You

OS/390
MVS Programming: Authorized
Assembler Services Reference, Volume 3
(LLACOPY-SDUMPX)

Publication No. GC28-1766-08

You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Today's date: _____

What is your occupation?

Newsletter number of latest Technical Newsletter (if any) concerning this publication:

How did you use this publication?

[ ]	As an introduction	[ ]	As a text (student)
[ ]	As a reference manual	[ ]	As a text (instructor)
[ ]	For another purpose (explain)		

Is there anything you especially like or dislike about the organization, presentation, or writing in this manual? Helpful comments include general usefulness of the book; possible additions, deletions, and clarifications; specific errors and omissions.

Page Number: Comment:

Name

Address

Company or Organization

Phone No.

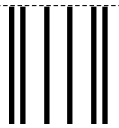
Reader's Comments — We'd Like to Hear from You
GC28-1766-08

Cut or Fold
Along Line

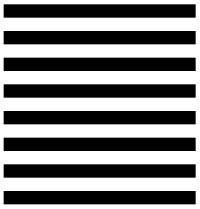
Fold and Tape

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

GC28-1766-08

Cut or Fold
Along Line

Program Number: 5647-A01

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC28-1766-08