Before using this information and the product it supports, be sure to read the general information under "Notices" on page [31]
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 22</td>
<td>BLRXSSP — Map the symbol service parameter list</td>
<td>115</td>
</tr>
<tr>
<td>Chapter 23</td>
<td>BLSUPPR2 — Map the expanded print service parameter list</td>
<td>117</td>
</tr>
<tr>
<td>Chapter 24</td>
<td>CALL — Pass control to a control section</td>
<td>119</td>
</tr>
<tr>
<td>Chapter 25</td>
<td>CHAP — Change dispatching priority</td>
<td>125</td>
</tr>
<tr>
<td>Chapter 26</td>
<td>CnzConv — Convert console name and ID</td>
<td>129</td>
</tr>
<tr>
<td>Chapter 27</td>
<td>CNMXURF — UCME look-up service macro</td>
<td>141</td>
</tr>
<tr>
<td>Chapter 28</td>
<td>CNZTRKR — Tracking interface macro</td>
<td>145</td>
</tr>
<tr>
<td>Chapter 29</td>
<td>CONVCON — Retrieve console information</td>
<td>149</td>
</tr>
<tr>
<td>Chapter 30</td>
<td>CONVTOD — Convert to time-of-day clock format</td>
<td>157</td>
</tr>
<tr>
<td>Chapter 31</td>
<td>CPOOL — Perform cell pool services</td>
<td>165</td>
</tr>
<tr>
<td>Chapter 32</td>
<td>CPUTIMER — Provide current CPU timer value</td>
<td>177</td>
</tr>
<tr>
<td>Chapter 33</td>
<td>CSRCESRV — Compress and expand data</td>
<td>181</td>
</tr>
<tr>
<td>Chapter 34</td>
<td>CSRCMPSC — Compress and expand data</td>
<td>187</td>
</tr>
<tr>
<td>Chapter 35</td>
<td>CSRC4ACT — Activate previously connected storage</td>
<td>193</td>
</tr>
<tr>
<td>Chapter 36</td>
<td>CSRC4BLD — Build a cell pool and initialize an anchor</td>
<td>197</td>
</tr>
<tr>
<td>Chapter 37</td>
<td>CSRC4CON — Connect cell storage to an extent</td>
<td>201</td>
</tr>
<tr>
<td>Chapter 38</td>
<td>CSRC4DAC — Deactivate an extent</td>
<td>205</td>
</tr>
<tr>
<td>Chapter 39</td>
<td>CSRC4DIS — Disconnect the cell storage for an extent</td>
<td>209</td>
</tr>
<tr>
<td>Chapter 40</td>
<td>CSRC4EXP — Expand a cell pool</td>
<td>213</td>
</tr>
<tr>
<td>Chapter 41</td>
<td>CSRC4FRE — Return a cell to a cell pool</td>
<td>219</td>
</tr>
<tr>
<td>Chapter 42</td>
<td>CSRC4FR1 — Return a cell to a cell pool</td>
<td>223</td>
</tr>
<tr>
<td>Chapter 43</td>
<td>CSRC4FR2 — Return a cell to a cell pool</td>
<td>227</td>
</tr>
<tr>
<td>Chapter 44</td>
<td>CSRC4GET — Allocate a cell from a cell pool</td>
<td>231</td>
</tr>
<tr>
<td>Chapter 45</td>
<td>CSRC4GT1 — Allocate a cell from a cell pool</td>
<td>235</td>
</tr>
<tr>
<td>Chapter 46</td>
<td>CSRC4GT2 — Allocate a cell from a cell pool</td>
<td>239</td>
</tr>
<tr>
<td>Chapter 47</td>
<td>CSRC4QCL — Query a cell</td>
<td>243</td>
</tr>
<tr>
<td>Chapter 48</td>
<td>CSRC4QEX — Query a cell pool extent</td>
<td>247</td>
</tr>
<tr>
<td>Chapter 49</td>
<td>CSRC4QPL — Query the cell pool</td>
<td>251</td>
</tr>
<tr>
<td>Chapter</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>76.</td>
<td>CSRREFR — Refresh an object</td>
<td>363</td>
</tr>
<tr>
<td>77.</td>
<td>CSRSAVE — Save changes made to a permanent object</td>
<td>367</td>
</tr>
<tr>
<td>78.</td>
<td>CSRSCOT — Save object changes in a scroll area</td>
<td>371</td>
</tr>
<tr>
<td>79.</td>
<td>CSRSI — System information service</td>
<td>375</td>
</tr>
<tr>
<td>80.</td>
<td>CSRUNIC — Unicode instruction services</td>
<td>393</td>
</tr>
<tr>
<td>81.</td>
<td>CSRVIEW — View an object</td>
<td>403</td>
</tr>
<tr>
<td>82.</td>
<td>CSVAPF — Query the list of APF-authorized libraries</td>
<td>407</td>
</tr>
<tr>
<td>83.</td>
<td>CSVINFO — Obtain information about loaded modules</td>
<td>419</td>
</tr>
<tr>
<td>84.</td>
<td>CSVQUERY — Contents supervisor query service</td>
<td>431</td>
</tr>
<tr>
<td>85.</td>
<td>DELETE — Relinquish control of a load module</td>
<td>445</td>
</tr>
<tr>
<td>86.</td>
<td>DEQ — Release a serially reusable resource</td>
<td>449</td>
</tr>
<tr>
<td>87.</td>
<td>DETACH — Detach a subtask</td>
<td>459</td>
</tr>
<tr>
<td>88.</td>
<td>DIV — Data-in-virtual</td>
<td>463</td>
</tr>
<tr>
<td>89.</td>
<td>DOM — Delete operator message</td>
<td>487</td>
</tr>
<tr>
<td>90.</td>
<td>DSPSERV — Create, delete, and control data spaces</td>
<td>491</td>
</tr>
<tr>
<td>91.</td>
<td>DSPSERV — Create, delete, and control hiperspaces</td>
<td>505</td>
</tr>
<tr>
<td>92.</td>
<td>EDTINFO — Obtain eligible device table information</td>
<td>519</td>
</tr>
<tr>
<td>93.</td>
<td>ENQ — Request control of a serially reusable resource</td>
<td>539</td>
</tr>
<tr>
<td>94.</td>
<td>ESPIE — Extended SPIE</td>
<td>553</td>
</tr>
<tr>
<td>95.</td>
<td>ESTAE and ESTAEX — Extended specify task abnormal exit</td>
<td>563</td>
</tr>
<tr>
<td>96.</td>
<td>EVENTS — Wait for one or more events to complete</td>
<td>577</td>
</tr>
<tr>
<td>97.</td>
<td>FREEMAIN — Free virtual storage</td>
<td>585</td>
</tr>
<tr>
<td>98.</td>
<td>GETMAIN — Allocate virtual storage</td>
<td>593</td>
</tr>
<tr>
<td>99.</td>
<td>GQSCAN — Extract information from global resource serialization queue</td>
<td>607</td>
</tr>
<tr>
<td>100.</td>
<td>HPSERV — Read from and write to a hiperspace</td>
<td>619</td>
</tr>
</tbody>
</table>

Appendices:
- Appendix. Accessibility: 629
- Notices: 631
- Index: 635
Figures

1. Sample User Parameter List for Callers in AR Mode ........................................... 5
2. Sample Macro Syntax Diagram ........................................................................... 13
3. Continuation Coding ......................................................................................... 15
4. CSRSIC from SYS1.SAMPLIB ........................................................................... 379
5. Return Code Area Used by DEQ ......................................................................... 454
6. Return Code Area Used by ENQ ........................................................................ 545
7. Creating a Table ................................................................................................. 580
8. Parameter List Format ....................................................................................... 581
9. Posting the Parameter List After ECBs 1 through 5 Processed and EVENTS WAIT= Issued 582
10. Posting the Parameter List While ECBs 1 through 5 Processed ............................ 583
11. Characteristics and Restrictions for Standard Hiperspaces ............................... 621
### Tables

1. Passing User Parameters in AR Mode ........................................... 5
2. Sample Callable Service Syntax Diagram .................................. 16
3. Service Summary ............................................................... 17
4. Return Codes for the CONVTOD Macro .................................. 160
5. Hexadecimal Return Codes for CPOOL LIST ............................. 172
6. Return and Reason Codes for the CPUTIMER Macro .................. 179
7. Return Codes for SERVICE=QUERY ........................................ 185
8. Return Codes for SERVICE=COMPRESS .................................. 185
9. Return Codes for SERVICE=EXPAND ..................................... 186
10. Return Codes for the CSRL16J Service ................................. 283
11. Return Codes for the CSRUNIC Macro .................................. 397
12. Return and Reason Codes for the CSVAPF Macro ..................... 411
13. Return Codes for the CSVINFO Macro .................................. 424
14. Return Codes for the DEQ Macro with the RET=HAVE Parameter .. 454
15. Return Codes for the ENQ Macro with the RET=TEST Parameter ... 545
16. Return Codes for the ENQ Macro with the RET=USE Parameter ..... 546
17. Return Codes for the ENQ Macro with the RET=CHNG Parameter ... 546
18. Return Codes for the ENQ Macro with the RET=HAVE Parameter ... 547
19. Return Codes for the FREEMAIN Macro ................................ 589
20. Return Codes for the GETMAIN Macro .................................. 602
21. Return Codes for the GQSCAN Macro .................................. 613
About this information

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

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

Who should use this information

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

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

Programmers using this information should have a knowledge of the computer, as described in Principles of Operation, as well as a knowledge of assembler language programming.

System macros require High Level Assembler. 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 information

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

Where to find more information

Where necessary, this publication references information in other documents, using shortened versions of the document title. For complete titles and order numbers of the documents for all products associated with z/OS, see z/OS Information Roadmap. The following table lists titles and order numbers for documents related to other products.

<table>
<thead>
<tr>
<th>Short Title Used in This Document</th>
<th>Title</th>
<th>Order Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principles of Operation</td>
<td>z/Architecture Principles of Operations</td>
<td>SA22-7832</td>
</tr>
</tbody>
</table>
Information updates on the web

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

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/Shelves/ZDOCAPAR

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

The z/OS Basic Skills Information Center

The z/OS Basic Skills Information Center is a Web-based information resource intended to help users learn the basic concepts of z/OS, the operating system that runs most of the IBM mainframe computers in use today. The Information Center is designed to introduce a new generation of Information Technology professionals to basic concepts and help them prepare for a career as a z/OS professional, such as a z/OS system programmer.

Specifically, the z/OS Basic Skills Information Center is intended to achieve the following objectives:

• Provide basic education and information about z/OS without charge
• Shorten the time it takes for people to become productive on the mainframe
• Make it easier for new people to learn z/OS.

To access the z/OS Basic Skills Information Center, open your Web browser to the following Web site, which is available to all users (no login required):

http://publib.boulder.ibm.com/infocenter/zos/basics/index.jsp
How to send your comments to IBM

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

Use one of the following methods to send us your comments:
1. Send an email to mhvrdfs@us.ibm.com
3. Mail the comments to the following address:
   IBM Corporation
   Attention: MHVRDFS Reader Comments
   Department H6MA, Building 707
   2455 South Road
   Poughkeepsie, NY 12601-5400
   U.S.A.
4. Fax the comments to us as follows:
   From the United States and Canada: 1+845+432-9405
   From all other countries: Your international access code +1+845+432-9405

Include the following information:
- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number:
  z/OS V1R13.0 MVS Assembler Services Reference ABE-H3P
  SA22-7606-13
- The topic and page number related to your comment
- The text of your comment.

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

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

If you have a technical problem

Do not use the feedback methods listed above. Instead, do one of the following:
- Contact your IBM service representative
- Call IBM technical support
- Visit the IBM support portal at [http://www.ibm.com/systems/z/support](http://www.ibm.com/systems/z/support)
Summary of changes

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

Changes made in z/OS Version 1 Release 13, as updated September 2012

This document contains information previously presented in z/OS MVS Programming: Assembler Services Reference ABE-HSP, SA22-7606-12, which supports z/OS Version 1 Release 13.

New information:
- Chapter 31, “CPOOL — Perform cell pool services,” on page 165 is updated for 1 MB pageable large pages.

Changes made in z/OS Version 1 Release 13

This document contains information previously presented in z/OS MVS Programming: Assembler Services Reference ABE-HSP, SA22-7606-11, which supports z/OS Version 1 Release 12.

Changed information:
- "ATTACHX—Create a new task" on page 57 is updated.
- Chapter 13, “BLSQMFLD — Specify a formatting model field,” on page 87 is updated.
- The minimum authorization of the ESPIE macro is updated. See Chapter 94, "ESPIE — Extended SPIE," on page 553.

Changes made in z/OS Version 1 Release 12

This document contains information previously presented in z/OS MVS Programming: Assembler Services Reference ABE-HSP, SA22-7606-10, which supports z/OS Version 1 Release 10.

Changed information:
- The description of user parameters that you can issue in AR mode is updated. See "User parameters" on page 4 in Chapter 1, “Using the services,” on page 1.
- The environment part of the ABEND macro is updated. See Chapter 2, “ABEND — Abnormally terminate a task,” on page 25.
- The description of the PARAM and PLIST8ARALETS parameters under the ATTACHX macro is updated. See "ATTACHX—Create a new task" on page 57.
- The description of the (addr) parameter under the CALL macro is updated. See Chapter 24, “CALL — Pass control to a control section,” on page 119.
- "Using the EVENTS macro" on page 579 is updated with steps to take after an ECB is posted when using the EVENTS macro.
• The "Readers' Comments - We'd Like to Hear from You" section at the back of this publication has been replaced. The hardcopy mail-in form has been replaced with a page that provides information appropriate for submitting readers comments to IBM.

**Deleted information:**

The RTLS function is withdrawn. The chapter on CSVRTLS — Define the RTLS Configuration is deleted.
Chapter 1. Using the services

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

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

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Service Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Compatibility of MVS macros&quot;</td>
<td>Macros</td>
</tr>
<tr>
<td>&quot;Addressing mode (AMODE)&quot; on page 2</td>
<td>Both</td>
</tr>
<tr>
<td>&quot;Address space control (ASC) mode&quot; on page 3</td>
<td>Both</td>
</tr>
<tr>
<td>&quot;ALET qualification&quot; on page 3</td>
<td>Both</td>
</tr>
<tr>
<td>&quot;User parameters&quot; on page 4</td>
<td>Macros</td>
</tr>
<tr>
<td>&quot;Telling the system about the execution environment&quot; on page 5</td>
<td>Macros</td>
</tr>
<tr>
<td>&quot;Specifying a macro version number&quot; on page 6</td>
<td>Macros</td>
</tr>
<tr>
<td>&quot;Register use&quot; on page 8</td>
<td>Both</td>
</tr>
<tr>
<td>&quot;Handling return codes and reason codes&quot; on page 8</td>
<td>Both</td>
</tr>
<tr>
<td>&quot;Handling program errors&quot; on page 9</td>
<td>Both</td>
</tr>
<tr>
<td>&quot;Handling environmental and system errors&quot; on page 10</td>
<td>Both</td>
</tr>
<tr>
<td>&quot;Using X-macros&quot; on page 11</td>
<td>Macros</td>
</tr>
<tr>
<td>&quot;Macro forms&quot; on page 11</td>
<td>Macros</td>
</tr>
<tr>
<td>&quot;Coding the macros&quot; on page 13</td>
<td>Macros</td>
</tr>
<tr>
<td>&quot;Coding the callable services&quot; on page 16</td>
<td>Callable Services</td>
</tr>
<tr>
<td>&quot;Including equate (EQU) statements&quot; on page 16</td>
<td>Callable Services</td>
</tr>
<tr>
<td>&quot;Link-editing linkage-assist routines&quot; on page 16</td>
<td>Callable Services</td>
</tr>
<tr>
<td>&quot;Service summary&quot; on page 17</td>
<td>Both</td>
</tr>
</tbody>
</table>

Compatibility of MVS macros

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

In most cases, the reverse is also true. When you assemble programs that issue macros on a particular version and release of MVS, those programs can run on earlier versions and releases of MVS, provided you request only those functions that are supported by the earlier version and release. This is useful for installations that write applications that might be assembled on one level of MVS, but run on a different level.
As MVS supports new architectures, addressability changes. To take best advantage of the new architectures, some macros have more than one possible expansion. You are required to have the macro expand according to the environment in which the program runs. This topic is described in this introductory information.

The problem of compatibility is not the same as selecting a macro version through the PLISTVER parameter to ensure the correct parameter list size for a macro. For selecting a parameter list version number, see "Specifying a macro version number" on page 6.

Addressing mode (AMODE)

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

In general,

• A program running in 24-bit addressing mode cannot pass parameters or parameter addresses that are higher than 16 megabytes. However, there are exceptions. For example, a program running in 24-bit addressing mode can:
  – Free storage above 16 megabytes using the FREEMAIN macro
  – Allocate storage above 16 megabytes using the GETMAIN macro
  – Use cell pool services for cell pools located in storage above 16 megabytes using the CPOOL macro
  – Use page services for storage locations above 16 megabytes using the PGSER macro
• A program running in 24-bit or 31-bit addressing mode cannot pass parameter addresses that are higher than 2 gigabytes, unless stated otherwise in the individual service description.
• If a program running in 31-bit or 64-bit addressing mode issues a service, parameters and parameter addresses can be above or below 16 megabytes, unless otherwise stated in the individual service description.

Some macros can generate code that is appropriate for programs in either 64-bit addressing mode or 24-bit or 31-bit addressing mode. These macros check a global symbol set by the SYSSTATE macro. See "Telling the system about the execution environment" on page 5 for more information.

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

You can invoke the following services in 64-bit addressing mode, subject to the “SVC or PC” restrictions mentioned later in this section, but you cannot pass parameters and parameter addresses above 2 gigabytes: ABEND, ATTACHX, CALLDISP, CHAP, CSVQUERY, DELETE, DEQ, DETACH, DOM, DSPSERV, DYNALLOC, ENQ, ESPIE, ESTAEX, EXCP, FREEMAIN, GETMAIN, GTRACE, IARVSERV, IDENTIFY, IEAARR, LINKX, LOAD, MODESET, PGSER, POST, RESERVE, SDUMPX, SETRP, STAX, STIMER, STIMERM, STORAGE, SYCHRX, TIME, TIMEUSED, TTIMER, VRADATA, WAIT, WTO, WTOR, and XCTL.
There are many services that support 64-bit addressing mode and parameter addresses above 2 gigabytes. Examples are IRAV64, IARST64, and ISGENQ. For details on the supported addressing mode and parameter address ranges for any specific service, see the following books:

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

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

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

For information about 64-bit addressing mode and the 64-bit GPR, see *z/OS MVS Programming: Assembler Services Guide*.

### Address space control (ASC) mode

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

Some macros can generate code that is appropriate for programs in either primary mode or AR mode. These macros check a global symbol set by the SYSSTATE macro. See "Telling the system about the execution environment" on page 5 for more information. Table 3 on page 17 lists the macros that check the global symbol.

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

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

### ALET qualification

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

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

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

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

The only ALETs that MVS services typically accept are:

- Zero (0), which specifies that the parameters are in the caller's primary address space
- An ALET for a public entry on the caller's dispatchable unit access list (DU-AL)
- An ALET for a common area data space (CADS)

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

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

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 the parameters whose addresses are in the parameter list. Control parameters control the operation of the macro itself. User parameters are parameters that a user provides to be passed through to a user routine. For example, the PARAM parameter on the ATTACHX macro defines user parameters. The ATTACHX macro passes these parameters to the routine that it attaches. All other parameters on the ATTACHX macro are control parameters that control the operation of the ATTACHX macro.

**Notes:**

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

The macros shown in Table 1 on page 5 allow a caller in AR mode to pass information in the form of a parameter list (or parameter lists) to another routine. This table identifies the parameter that receives the ALET-qualified address of the parameter list and tells you where the target routine finds the ALET-qualified address.
Table 1. Passing User Parameters in AR Mode

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

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

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

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

![Figure 1. Sample User Parameter List for Callers in AR Mode](image)


**Telling the system about the execution environment**

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

- The addressing mode (AMODE) at the time the macro is issued
- The ASC mode of the program at the time the macro is issued
- The Architectural level in which the program runs
For macros that are sensitive to their environment, use the SYSSTATE macro to define the environment. During the assembly stage, SYSSTATE sets one or more global symbols. Later, in your source code, the macro checks the global symbols and generates the correct code, which might mean avoiding using a z/Architecture® instruction or an access register. Table 3 on page 17 lists MVS macros and identifies macros that need to know the environmental characteristics.

**IBM recommends** you issue the SYSSTATE macro before you issue other macros. Once a program has issued SYSSTATE, there is no need to reissue it, unless the program switches from one AMODE to another or one ASC mode to another or has code paths that are isolated according to architecture level or operating system release. If you switch AMODE or ASC mode to a different architecture code path, issue SYSSTATE immediately after the switch to indicate the new state. In general, specify SYSSTATE ARCHLVL=1, and switch to SYSSTATE ARCHLVL=2 before issuing macros in sections of code that run in z/Architecture mode. If you do not issue the SYSSTATE macro, the system assumes the macro is issued:

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

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

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Parameter on SYSSTATE</th>
<th>Global symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>AMODE of 64-bit, or 24-bit or 31-bit</td>
<td>AMODE64=YES or NO</td>
<td>&amp;SYSAM64</td>
</tr>
<tr>
<td>Primary or AR ASC mode</td>
<td>ASCENV=P or AR</td>
<td>&amp;SYSASCE</td>
</tr>
<tr>
<td>Architectural level of z/Architecture</td>
<td>ARCHLVL=0, 1 or 2</td>
<td>&amp;SYSALVL</td>
</tr>
</tbody>
</table>

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

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

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

**Specifying a macro version number**

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

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

How to request a macro version using PLISTVER

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

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

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

Hints for using PLISTVER

There are some general considerations that you should keep in mind when specifying the version of a macro with PLISTVER:
- If PLISTVER is omitted, the macro generates a parameter list of the lowest version that allows all the parameters specified to be processed.
- If you code PLISTVER=n and then specify any version ‘n+1’ parameter, the macro will not assemble.
- If you code PLISTVER=n and do not specify any version ‘n’ parameter, the macro will generate a version ‘n’ parameter list.
- If you are using the standard form of the macro (MF=S), there is no reason you need to code the PLISTVER parameter.
- Not all macros have the same version numbers. The version numbers need not be contiguous.

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

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

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

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

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

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

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

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

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

Handling return codes and reason codes

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

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

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

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

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

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

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

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

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

**Handling program errors**

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

1. Breaking one of the rules of the service. For example:
   • Passing parameters that are either in the wrong format or not valid
   • Violating one of the environment requirements (addressing mode, locking
     requirements, dispatchable unit mode, and so on)
   • Providing insufficient storage for information to be returned by the system.

2. Causing errors related to the parameter list. For example:
   • Coding an incorrect combination of parameters
   • Coding one or more parameters on the service incorrectly
   • Inadvertently overlaying an area of the parameter list storage
   • Inadvertently destroying the pointer to the parameter list.

3. Requesting a service or function for which the calling program is not authorized,
or which is not available on the system on which the program is running.

In each of the first two cases, you can correct your program. For completeness, the
return and reason code descriptions give you specific actions to perform, even
when it might seem obvious what the action should be.
In the third case, you might have to contact your system administrator or system programmer to obtain the necessary authorization, or to request that the service or function be made available on your system, and the return or reason code description asks you to take that step.

**Note:** Generally, the system does not take dumps for errors that your program causes when issuing a system service. If you require such a dump, then it is your responsibility to request one in your recovery routine. See the section on providing recovery in [z/OS MVS Programming: Assembler Services Guide](https://www.ibm.com) for information about writing recovery routines.

### Handling environmental and system errors

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

- The request being made through the service exceeds some internal system limit. Sometimes, rerunning your program or retrying the request results in successful completion. If the problem persists, it might be an indication of a larger problem requiring you to consult your system programmer, or possibly IBM support personnel. Your system programmer might be able to tune the system or cancel users so that the limit is no longer exceeded.

- The request exceeds an installation-defined limit. If the problem persists, the action might be to contact your system programmer and request that a specification in an installation exit or parmlib member be modified.

- The system cannot obtain storage, or some other resource, for your request. If the problem persists, the action might be to check with the operator to see if another user in the installation is causing the problem, or to see if the entire installation is experiencing storage constraint problems.

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

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

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

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

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

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

The X-macros described in this book are:
- ATTACHX
- ESTAEX
- LINKX
- SNAPX
- SYNCHX
- XCTLX

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

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

- Callers in primary mode can invoke either macro.

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

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

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

**IBM recommends** you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, in which case, you should use ESTAE.

**Macro forms**

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

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

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

### 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 diagram near the beginning of the macro description. The diagram shows how to code the macro. The syntax diagram does not explain the meanings of the parameters; the meanings are explained in the parameter descriptions that follow the syntax diagram.

The syntax tables assume that the standard begin, end, and continue columns are used. Thus, column 1 is assumed as the begin column. To change the begin, end, and continue columns, use the ICTL instruction to establish the coding format you want to use. If you do not use ICTL, the assembler recognizes the standard columns. To code the ICTL instruction, see [HLASM Language Reference](#).

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

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>name: symbol. Begin name in column 1.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b One or more blanks must precede TEST.</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>TEST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b One or more blanks must follow TEST.</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>MATH</td>
<td></td>
</tr>
<tr>
<td></td>
<td>HiST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GEOG</td>
<td></td>
</tr>
<tr>
<td></td>
<td>,DATA= data addr</td>
<td>data addr: RX-type address, or register (2) - (12)</td>
</tr>
<tr>
<td>B1</td>
<td>,LNG= data length</td>
<td>data length: symbol or decimal digit, with a maximum value of 256.</td>
</tr>
<tr>
<td></td>
<td>,FMT=HEX</td>
<td>Default: FMT=HEX</td>
</tr>
<tr>
<td></td>
<td>,FMT=DEC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>,FMT=BIN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>,PASS= value</td>
<td>value: symbol, decimal digit, or register (1) or (2) - (12). Default: PASS=65</td>
</tr>
<tr>
<td></td>
<td>,grade</td>
<td>grade: symbol, decimal digit, or register (1) or (2) - (12).</td>
</tr>
</tbody>
</table>

**Figure 2. Sample Macro Syntax Diagram**
Column A and Column B contain those parameters that are allowed for the macro. Column A contains those parameters that are required; column B contains those parameters which are optional.

If a single line appears, as shown in A1 and B1, then that is the only available choice for the particular parameter.

If two or more lines appear together, as shown in A2 and B2, the parameters on those lines are mutually exclusive, that is, you can code any one of those parameters.

A further distinction is made between mandatory and optional parameters. The parameter descriptions that follow the syntax table clearly identify those parameters which are optional.

The third column, C, provides additional information about coding the macro.

When substitution of a variable is required in column C, the following classifications are used:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Symbol</td>
<td>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.</td>
</tr>
<tr>
<td>Decimal digit</td>
<td>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.</td>
</tr>
<tr>
<td>Register (2)-(12)</td>
<td>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.</td>
</tr>
<tr>
<td>Register (0)</td>
<td>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.</td>
</tr>
<tr>
<td>Register (1)</td>
<td>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.</td>
</tr>
<tr>
<td>Register (15)</td>
<td>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.</td>
</tr>
<tr>
<td>RX-type address</td>
<td>Any address that is valid in an RX-type instruction (for example, LA).</td>
</tr>
<tr>
<td>RS-type address</td>
<td>Any address that is valid in an RS-type instruction (for example, STM).</td>
</tr>
<tr>
<td>RS-type name</td>
<td>Any name that is valid in an RS-type instruction (for example, STM).</td>
</tr>
</tbody>
</table>
A-type address

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

Default

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

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

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

Continuation lines

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

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

You can code the parameter field being continued in one of two ways. Code the parameter field through column 71, with no blanks, and continue in column 16 of the next line; or truncate the parameter field by a comma, where a comma normally falls, with at least one blank before column 71, and then continue in column 16 of the next line. Figure 3 shows an example of each method.

Figure 3. Continuation Coding
Coding the callable services

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

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

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

**Table 2. 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:
Note: Omitting NCAL from the linkedit parameters (as the example shows) and specifying SYS1.CSSLIB in the //SYSLIB statement, as shown, causes the addresses of all required linkage-assist routines to be automatically resolved. This statement saves you the trouble of having to specify individual linkage-assist routines in INCLUDE statements.

Service summary

Table 3 on page 17 lists services described in the following:

- z/OS MVS Programming: Assembler Services Reference ABE-HSP
- z/OS MVS Programming: Assembler Services Reference IAR-XC7

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

Notes:
1. A program running in primary ASC mode when PASN=HASN=SASN can issue any of the services listed in the table.
2. Cross memory mode means that at least one of the following conditions is true:
   - PASN≠SASN The primary address space (PASN) and the secondary address space (SASN) are different.
   - PASN≠HASN The primary address space (PASN) and the home address space (HASN) are different.
   - SASN≠HASN The secondary address space (SASN) and the home address space (HASN) are different.

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

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

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABEND</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ALESERV</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ASASYMBM</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTACH</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ATTACHX</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BLDMPB</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>BLSABDPL</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSACBSP</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSADSY</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSAPCQE</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSQFXL</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSQMDEF</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSQMFLD</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSQSHDR</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRDRPX</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRESSY</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRNMPP</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRPRD</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRPWHS</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRASBY</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRXMSP</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSRXSSP</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>BLSUPPR2</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CALL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CHAP</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CNZCONV</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>CNZTRKR</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CONVCON</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CONVTOD</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CPOOL</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CPUUTIMER</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CSRCESRV</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CSRCMPSC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CSREVW</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRIDAC</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSL16J</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSPACT</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPLBD</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPCON</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPDAC</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPDIS</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSRPEXP</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPFRE</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPFRR1</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPGET</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPGT1</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPGT</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRPGT1</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRREFR</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRSAVE</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRSCEOT</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSRSQI</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CSRUNIC</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CSRVIEW</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>CSVAPF</td>
<td>Yes (See note 7 on page 23)</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>CSVINFO</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>CSVQUERY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>DELETE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DEQ</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DETACH</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DIV</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>DOM</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>DSPSERV</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>EDTINFO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ENQ</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ESPIE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>ESTAE (See note 2 on page 23)</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ESTAEX</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>EVENTS</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>FREEMAIN</td>
<td>No (See note 3 on page 23)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>GETMAIN</td>
<td>No (See note 3 on page 23)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>GQSCAN</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>HSPSERV</td>
<td>Yes</td>
<td>Yes (See note 4 on page 23)</td>
<td>(See note 5 on page 23)</td>
<td>No</td>
</tr>
<tr>
<td>IARCP64</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IARR2V</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IARST64</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IARVSERVER</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IARV64</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IDENTIFY</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEAARR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEABRC</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>IEAINTKN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEALSRBY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEAMETR</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IEANTCR</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>IEANTDL</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>IEANTRT</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>IVEAPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAPE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEVDPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEVDPE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVPSE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVPSE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVRLS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVRLS2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVRPI</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVRPI2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVTPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVXFR</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IVEAVXFR2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEA4APE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4APE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4DPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4DPE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4PSE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4PSE2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4RLS</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4RLS2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4RPI</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEA4RPI2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4TPE</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4XFR</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEA4XFR2</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>IEFDDSRV</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IEFSSI</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IOCINFO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IOSCHPD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>ITZEVENT</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>ITZQUERY</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>IXGBRWSE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IXGCONN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IXGDELET</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>IXGIMPRT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IXGINVNT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IXGOFFLD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IXGQUERY</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IXGUPDAT</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>IXGWRITE</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LINK</td>
<td>Yes (See note [1] on page 22)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LINKX</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LOAD</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>LSEXTPAND</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGLOAD</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGOUT</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGRLSE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>PGSER</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>POST</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>QRYLANG</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>REFPAT</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>RESERVE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>RETURN</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SAVE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>SETRP</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SNAP</td>
<td>Yes (See note [1] on page 22)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SNAPX</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SPIE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
Table 3. Service Summary (continued)

<table>
<thead>
<tr>
<th>Service</th>
<th>Can be issued in AR ASC mode</th>
<th>Can be issued in cross memory mode</th>
<th>Checks SYSSTATE</th>
<th>Can be issued in 64-bit AMODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPLEVEL</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>STAE</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>STATUS</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>STCKCONV</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>STCKSYNC</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>STIMER</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>STIMERM</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>STORAGE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>SYMRBLD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYMREC</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYNCH</td>
<td>Yes (See note 1 on page 22)</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>SYNCHX</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>SYSSTATE</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TCBTOKEN</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TESTART</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TIME</td>
<td>Yes (See note 6 on page 23)</td>
<td>Yes (See note 6 on page 23)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TIMEUSED</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TRANMSG</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>TTIMER</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>UCBDEVN</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>UCBINFO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UCBCAN</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>UPDTEMPB</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>VRADATA</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>WAIT</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>WTL</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>WTO</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>WTOR</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>XCTL</td>
<td>Yes (See note 1 on page 22)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XCTLX</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

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

IBM recommends that all callers in AR mode use the X-macros (ATTACHX, LINKX, SNAPX, SYNCHX, and XCTLX). If a program in AR mode issues ATTACH, LINK, SNAP, SYNCH, or XCTL after issuing SYSSTATE ASCENV=AR, the system substitutes the corresponding X-macro and issues a message telling you that it made the substitution.

2. The only programs that can use ESTAE are programs that are in primary mode with PASN=HASN=SASN. Callers in AR mode or in cross memory mode must use ESTAEX instead of ESTAE.

IBM recommends you always use ESTAEX unless your program and your recovery routine are in 24-bit addressing mode, in which case, you should use ESTAE.

3. Problem state AR mode callers must use the STORAGE macro instead of using GETMAIN or FREEMAIN.

4. PASN=HASN=SASN for a non-shared standard hiperspace for which an ALET is not used (the HSPALET parameter is omitted).

5. If you use the HSPALET parameter, the HSPSERV macro checks SYSSTATE.

6. Only TIME LINKAGE=SYSTEM can be issued in AR mode, and can be issued in cross memory mode. TIME LINKAGE=SVC cannot be issued in AR mode or in cross memory mode.

7. For the QUERY request, CSVAPF can be issued only in primary mode. For all other requests, CSVAPF can be issued in primary or AR mode.
Chapter 2. ABEND — Abnormally terminate a task

The ABEND macro is used to initiate error processing for a task. ABEND can request a full or tailored dump of virtual storage areas and control blocks pertaining to the tasks being abnormally terminated, and can specify that the entire job step is to be abnormally terminated. If a user-written recovery routine was activated at the time the ABEND macro was issued, it will get control before the task is terminated. This routine may recover the task and allow it to retry. See z/OS MVS Programming: Assembler Services Guide for information on how to provide user-written recovery routines.

If the job step task is abnormally terminated or if ABEND specifies job step termination, the completion code is recorded on the system output device, and the remaining job steps in the job are either skipped or executed as specified in their job control statements.

If the job step is not to be terminated, the system takes the following actions:
• It terminates the task that was active when ABEND was issued and all of the subtasks of that active task.
• It posts the completion code as indicated in the completion code parameter description below.
• It selects the end-of-task exit routine specified in the ATTACH macro to receive control. That end-of-task routine created the task that issued ABEND. The system gives the exit routine control when the originating task of the task for which ABEND was issued becomes active. It does not give control to any of the end-of-task exit routines specified for any subtasks of the task for which ABEND was issued.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31- or 64-bit
ASC mode: Primary, secondary, or access register (AR)
Interrupt status: Enabled or disabled for I/O and external interrupts
Locks: No locks required
Control parameters: None.

Programming requirements

If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before you issue the ABEND macro. SYSSTATE ASCENV=AR tells the ABEND macro to generate code appropriate for AR mode.

Restrictions

None.
**ABEND Macro**

**Input register information**

Before issuing the ABEND 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**

None, because control does not return to the caller.

**Performance implications**

None.

**Syntax**

The ABEND macro is written as follows:

```
name name
\bslash

ABEND
\bslash

comp code comp code
,REASON=reason code
,DUMP
,,STEP
,,code type
,DUMP,,code type
,,STEP,code type
,DUMP,STEP,,code type
,DUMP,STEP,code type
,DUMP,DUMPOPT=parm list addr
,DUMP,DUMPOPX=parm list addr
```

**Parameters**

The parameters are explained as follows:

- **comp code**
  Specifies the completion code associated with the abnormal termination. If the job step is to be terminated, the decimal representation of the user completion...
code or the hexadecimal representation of the system completion code is recorded on the system output device. If the job step is not to be terminated, the completion code is placed in the TCB of the active task, and in the ECB specified in the ECB parameter of the ATTACH macro issued to create the active task. If you specify a hexadecimal digit, you must use X’d’d’ format to distinguish the hexadecimal from decimal.

\texttt{,REASON=reason code}

Specifies the \textit{reason code} that the user wants to pass to subsequent recovery exits. The value range for the \textit{reason code} is a 32-bit hexadecimal number or a 31-bit decimal number. This \textit{reason code} supplements the completion code associated with an abnormal termination, allowing the user to uniquely identify the cause of the abnormal termination. The \textit{reason code} is propagated to each recovery exit.

\texttt{,DUMP}
\texttt{,STEP}
\texttt{,code type}
\texttt{,DUMP,STEP}
\texttt{,DUMP,code type}
\texttt{,STEP,code type}
\texttt{,DUMP,STEP,code type}
\texttt{,DUMP,DUMPOPT=parm list addr}
\texttt{,DUMP,DUMPOPX=parm list addr}

Specifies options available with the ABEND macro:

DUMP specifies that a dump is requested of virtual storage areas assigned to the task and control blocks pertaining to the task. A separate dump is provided for each of the tasks being terminated as a result of ABEND. If a //SYSABEND, //SYSMDUMP, or //SYSUDUMP DD statement is not provided, the DUMP parameter is ignored.

For z/OS UNIX System Services, the system writes a core dump, which is a SYSDUMP to an HFS file, for errors following an exec or fork() function when the original address space had a SYSDUMP DD statement. For more information, see \textit{AD/Cycle® LE/370 Debugging and Run-Time Messages Guide}.

STEP specifies that the entire job step of the active task is to be abnormally terminated.

\textit{Note:} If the STEP parameter is coded in an ABEND macro under TSO, the TSO job will be terminated.

\textit{code type} specifies that the completion code is to be treated as a USER or SYSTEM code.

DUMPOPT and DUMPOPX specify the address of a parameter list of options for a tailored dump. To create the parameter list, use the list form of either the SNAP or SNAPX macro, or code data constants in your program. DUMPOPT specifies the address of a parameter list that the SNAP macro created. DUMPOPX specifies the address of a parameter list that the SNAPX macro created.

The TCB, DCB, ID, and STRHDR options available on SNAP will be ignored if they appear in the parameter list; the TCB used will be that of the task being terminated, the DCB used will be provided by the ABDUMP routine. If a //SYSABEND, //SYSMDUMP, or //SYSUDUMP DD statement is not provided, this parameter is ignored.
ABEND Macro

If the dump options specified include ranges of storage areas to be dumped, only the storage areas in the first thirty ranges will be dumped. If SUBPLST is specified in the SNAP or SNAPX parameter list passed to the ABEND macro via DUMPOPT or DUMPOPX, the first seven subpools will be dumped.

The dump option parameter list, storage ranges, and subpools must be in the primary address space.

ABEND codes

None.

Return and reason codes

None.

Example 1

Terminate with a user completion code of 432.
ABEND 432

Example 2

Terminate with the user completion code that is contained in register 5. The entire job step is to be terminated.
ABEND (5),,STEP

Example 3

Terminate with a system completion code of X'0C4'.
ABEND X'0C4',,SYSTEM
Chapter 3. ALESERV — Control entries in the access list

Description

The ALESERV macro manages the contents of access lists. An access list is a table in which each entry identifies an address space, data space, or hiperspace to which a program (or programs) has access. Access list entry tokens (ALETs) index the entries in the access list. Use the ALESERV macro to:

- Add an entry to a DU-AL for an address space, data space, or nonshared standard hiperspace (ADD parameter)
- Add an entry for the primary address space to the DU-AL (ADDPASN parameter)
- Add an entry for a SCOPE=SINGLE data space to the PASN-AL.
- Delete an entry from a DU-AL (DELETE parameter)
- Obtain a STOKEN for a specified ALET (EXTRACT parameter)
- Locate an ALET for a specified STOKEN (SEARCH parameter)
- Obtain the STOKEN of the home address space (EXTRACTH parameter).

A problem state program can use ALESERV to create an entry associated with an address space only if it is running with an appropriate extended authorization index (EAX) value. To set up EAX-authorization, a program must be in supervisor state. Information on EAX-authorization appears in the books that are available to system programmers who write programs in supervisor state.

On the ALESERV macro, address spaces, data spaces, and hiperspaces are identified through STOKENs, an identifier similar to an address space identifier (ASID).

For information about access lists, ALETs, data spaces, and hiperspaces, see appropriate chapters in the book "z/OS MVS Programming: Assembler Services Guide." That book contains many examples of using ALESERV.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with any PSW key.
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31- bit
- **ASC mode**: Primary or access register (AR)
- **Interrupt status**: Enabled for I/O and external interrupts for ADD, ADDPASN, and DELETE requests. Enabled or disabled for I/O and external interrupts for requests other than ADD, ADDPASN, and DELETE
- **Locks**: No locks held for ADD, ADDPASN, and DELETE requests. For requests other than ADD, ADDPASN, and DELETE, the caller may hold locks, but is not required to hold any.
- **Control parameters**: Can reside in any addressable area

Programming requirements

For ADD and DELETE requests, the caller of the ALESERV macro must be one of the following:

- The owner or creator of the data space
ALESERV Macro

- The owner of the hiperspace.

Restrictions

None.

Input register information

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

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code associated with the return code for SEARCH and EXTRACT requests; otherwise, used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Address of the ALESERV parameter list</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the access registers contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>ALET for the parameter list</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 ALESERV macro is written as follows:

```
name name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede ALESERV.

ALESERV

b

One or more blanks must follow ALESERV.
```
Valid parameters (required parameters are underlined):

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters Required</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>AL, STOKEN, ACCESS, ALET, CHKPT, RELATED</td>
<td>ACCESS=PUBLIC</td>
</tr>
<tr>
<td>ADDPASN</td>
<td>ALET, CHKPT, RELATED</td>
<td>AL=WORKUNIT</td>
</tr>
<tr>
<td>DELETE</td>
<td>ALET, RELATED</td>
<td></td>
</tr>
<tr>
<td>EXTRACT</td>
<td>ALET, STOKEN, RELATED</td>
<td></td>
</tr>
<tr>
<td>SEARCH</td>
<td>ALET, STOKEN, AL, RELATED</td>
<td></td>
</tr>
<tr>
<td>EXTRACTH</td>
<td>STOKEN, RELATED</td>
<td></td>
</tr>
</tbody>
</table>

ALET=alet-addr: RX-type address or register (2) - (12).

STOKEN=stoken-addr: RX-type address.

Note: If you specify register notation, the register contains the ALET, rather than the address of the ALET.

Parameters
The parameters are explained as follows:

ADD
Requests that the system add an entry to the access list. You are required to use two parameters:
- STOKEN specifies the address space, data space, or hiperspace that the entry represents
- ALET specifies the address of the location where the system returns the ALET.

For access list entries that represent an address space, you can also specify whether an entry is public or private (ACCESS parameter). To add an entry for an address space, the caller must have EAX-authority to the target address space.

For access list entries that represent a data space or hiperspace, the entry must be public.

A problem state program can add an entry for a SCOPE=SINGLE data space to the PASN-AL if both of the following are true:
- The caller owns or created the data space.
- An entry for the data space is not already on the PASN-AL through the action of another problem state program.

ADDPASN
Requests that the system add an entry for the primary address space to the
ALESERV Macro

DU-AL without requiring a user to have EAX-authority to the address space. ALET, required with ADDPASN, receives the ALET that indexes into the entry. The entry is a public entry.

DELETE
Requests that the system delete an entry from the DU-AL. ALET, required with DELETE, identifies the entry to be deleted.

EXTRACT
Requests that the system find the STOKEN of the specified ALET. The caller can obtain the STOKEN for any address space, data space, or hiperspace that is represented by a valid entry on the DU-AL or PASN-AL. ALET and STOKEN are required parameters.

SEARCH
Requests that the system search through the DU-AL or PASN-AL for an ALET that corresponds to a specified STOKEN. ALET and STOKEN are required parameters. AL is an optional parameter; AL=DU-AL is the default.

EXTRACTH
Requests that the system find the STOKEN of the home address space. STOKEN is a required parameter.

\ACCESS=PUBLIC
\ACCESS=PRIVATE
Specifies whether the access list entry you are adding is public or private. You cannot add a private entry for a data space or hiperspace.

\AL=WORKUNIT
\AL=PASN
Specifies whether the access list is a DU-AL (WORKUNIT) or a PASN-AL (PASN). For the ADD request, AL identifies the type of access list.

For the SEARCH request, AL specifies whether the system is to search through the DU-AL or the PASN-AL.

\ALET=alet-addr
Specifies the 4-byte ALET that either you provide or the system returns, depending on the other parameters you specify on ALESERV. When you use RX-type notation, alet-addr specifies the address of the 4-byte field that contains the ALET. When you use register notation, alet-addr specifies a register that contains the ALET itself, rather than the address of the ALET.

For the ADD and ADDPASN requests, the system returns the ALET of the added entry.

For the DELETE request, you provide the ALET for the access list entry to be deleted. Do not specify an ALET of 0, 1, or 2.

For the EXTRACT request, you provide the ALET whose STOKEN you require. The system returns the STOKEN in stoken-addr.

For the SEARCH request, you specify where in the access list the system is to begin the search:
• If you specify minus one (-1), the system starts searching at the beginning of the DU-AL or PASN-AL.
• If you specify a valid ALET, the system starts searching with the next ALET in the access list.

The system then returns the searched-for ALET, if present. Otherwise, alet-addr is unchanged and register 15 contains a return code that specifies that an ALET for the STOKEN is not on the access list.
## ALESERV Macro

\textbf{\texttt{STOKEN}=	extit{token-addr}}

Specifies the 8-byte identifier of an address space, data space, or hiperspace. For the ADD request, STOKEN identifies the space that the program wants to access.

For the EXTRACT request, the system returns the STOKEN that corresponds to the specified ALET.

For the SEARCH request, STOKEN identifies the STOKEN for which the system is to return the corresponding ALET.

For the EXTRACTH request, the system returns the STOKEN of the home address space.

\textbf{\texttt{CHKPT}=FAIL,\texttt{CHKPT}=IGNORE}

Specifies how the system is to process a checkpoint request made through the CHKPT macro, in relation to the access list entry being added. If you specify CHKPT=IGNORE, the system ignores the access list entry added (DU-AL or PASN-AL) and processes the checkpoint operation. If you specify CHKPT=FAIL, the system rejects the checkpoint operation. The default is CHKPT=FAIL.

If you specify CHKPT=IGNORE, you assume full responsibility of managing the data space or nonshared standard hiperspace storage. See \textit{z/OS MVS Programming: Assembler Services Guide} for more information on using checkpoints with data spaces and hiperspaces.

\textbf{\texttt{RELATED}=\textit{any-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 and reason codes

When control is returned from ALESERV ADD, register 15 contains one of the following hexadecimal return codes. A return code of 8 or greater means the system rejects the request.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** ALESERV ADD has completed successfully.  
                       | **Action:** None. |
| 8                       | **Meaning:** Program error. The caller was not EAX-authorized to the specified space. The entry is not added.  
                       | **Action:** Verify that the intended STOKEN is specified. |
| 0C                      | **Meaning:** Environmental error. The current access list cannot be expanded. There are no free access list entries and the maximum size has been reached.  
<pre><code>                   | **Action:** Delete unused entries and reissue the request. |
</code></pre>
<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td><strong>Meaning:</strong> Environmental error. ALESERV could not obtain storage for an expanded access list. <strong>Action:</strong> Retry the request.</td>
</tr>
<tr>
<td>18</td>
<td><strong>Meaning:</strong> Program error. The caller in problem state with PSW key 8 - F tried to add an entry to the PASN-AL for a space other than a SCOPE=SINGLE data space. <strong>Action:</strong> Change the request to add the data space as SCOPE=SINGLE or change your program to run in supervisor state or key 0 - 7.</td>
</tr>
<tr>
<td>1C</td>
<td><strong>Meaning:</strong> Program error. The caller is holding a lock. <strong>Action:</strong> Release all locks before calling ALESERV.</td>
</tr>
<tr>
<td>20</td>
<td><strong>Meaning:</strong> Program error. The caller is disabled. <strong>Action:</strong> Enable your program before it issues ALESERV.</td>
</tr>
<tr>
<td>24</td>
<td><strong>Meaning:</strong> Program error. AR 1 contained an ALET of 1 on input, or a PASN-AL ALET. <strong>Action:</strong> Verify that AR 1 contains either an ALET of 0 or the ALET for the caller's DU-AL.</td>
</tr>
<tr>
<td>38</td>
<td><strong>Meaning:</strong> Program error. The input STOKEN is not valid. <strong>Action:</strong> Verify that the specified STOKEN is a valid STOKEN.</td>
</tr>
<tr>
<td>4C</td>
<td><strong>Meaning:</strong> Program or environmental error. The space represented by the input STOKEN is not valid for cross memory access. <strong>Action:</strong> None required. However, you may want to take some action based upon your application.</td>
</tr>
<tr>
<td>50</td>
<td><strong>Meaning:</strong> Program error. The ALESERV parameter list is not valid. <strong>Action:</strong> Verify that your program is not overwriting the parameter list and that the execute form of the macro correctly addresses the parameter list.</td>
</tr>
<tr>
<td>54</td>
<td><strong>Meaning:</strong> Program error. The caller tried to add a data space or hiperspace to an access list as a private entry. <strong>Action:</strong> Specify ACCESS=PUBLIC instead of ACCESS=PRIVATE.</td>
</tr>
<tr>
<td>5C</td>
<td><strong>Meaning:</strong> Program error. The caller tried to add a data space or a hiperspace to an access list without proper authority. <strong>Action:</strong> Correct your program to specify STOKENs for spaces for which your program is authorized.</td>
</tr>
<tr>
<td>60</td>
<td><strong>Meaning:</strong> System error. An unexpected error occurred. The request was not completed. <strong>Action:</strong> Retry the request.</td>
</tr>
<tr>
<td>62</td>
<td><strong>Meaning:</strong> Program error. A previous error in your program left the access list in an unexpected format. The error might have occurred because the SRB environment was not valid when the system dispatched an SRB. The system did not perform the ALESERV ADD request. <strong>Action:</strong> Determine the cause of the error that preceded the ALESERV ADD request. Correct the error and rerun the program.</td>
</tr>
</tbody>
</table>
### ALESERV Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 64                      | **Meaning:** Program error. A problem-state caller with PSW key 8 - F tried to add an entry using CHKEAX=NO.  
**Action:** Specify CHKEAX=YES. |
| 68                      | **Meaning:** Program error. The caller attempted to add a hiperspace that is not a nonshared standard hiperspace owned by the caller.  
**Action:** Verify that the options specified on your ADD request do not violate the rules for adding entries for hiperspaces to access lists. |
| 6C                      | **Meaning:** Program error. The caller tried to add an entry for a SCOPE=COMMON data space to a DU-AL.  
**Action:** Change your program to request the ADD to be made to the PASN-AL. |
| 70                      | **Meaning:** Environmental error. The caller attempted to add a hiperspace to an access list.  
**Action:** Modify your program to use the HPSERV macro to access the data in the hiperspace. |
| 74                      | **Meaning:** Program error. A problem state program with PSW key 8 - F has already added an entry for the data space to the PASN-AL and the entry still exists.  
**Action:** Change your program's logic so that it does not request the second ADD. |
| 78                      | **Meaning:** Program error. A problem state program with PSW key 8 - F tried to add an entry to the PASN-AL. The program is neither the owner nor the creator of the data space.  
**Action:** Change your program's logic so that it does not add a data space it did not create or does not own. |
| 80                      | **Meaning:** Program error. The caller attempted to add a subspace access list entry to the PASN-AL.  
**Action:** Change the request to add the subspace access list entry to the DU-AL. |
| 84                      | **Meaning:** Program error. The caller tried to add a subspace access list entry to the DU-AL, but the caller is not running under the task that owns the subspace.  
**Action:** Ensure that your program is running under the task that created the subspace, or check that you are supplying the correct STOKEN. |

When control is returned from ALESERV ADDPASN, register 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** ALESERV ADDPASN has completed successfully.  
**Action:** None. |
### ALESERV Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| C                       | **Meaning:** Environmental error. The DU-AL cannot be expanded. There are no free ALEs, and the maximum size has been reached.  
**Action:** Delete unused entries and reissue the request. |
| 10                      | **Meaning:** Environmental error. ALESERV could not obtain storage for an expanded access list.  
**Action:** Retry the request. |
| 1C                      | **Meaning:** Program error. The caller is holding a lock.  
**Action:** Release all locks before calling ALESERV. |
| 20                      | **Meaning:** Program error. The caller is disabled.  
**Action:** Enable your program before it issues ALESERV. |
| 24                      | **Meaning:** Environmental error. AR 1 contained an ALET of 1 on input, or a PASN-AL ALET.  
**Action:** Verify that AR 1 contains either an ALET of 0 or the ALET for the caller’s DU-AL. |
| 50                      | **Meaning:** Program error. The ALESERV parameter list is not valid.  
**Action:** Verify that your program is not overwriting the parameter list and that the execute form of the macro correctly addresses the parameter list. |
| 60                      | **Meaning:** System error. An unexpected error occurred. The request was not completed.  
**Action:** Retry the request. |
| 62                      | **Meaning:** Program error. A previous error in your program left the access list in an unexpected format. The error might have occurred because the SRB environment was not valid when the system dispatched an SRB. The system did not perform the ALESERV ADDPASN request.  
**Action:** Determine the cause of the error that preceded the ALESERV ADD request. Correct the error and rerun the program. |

When control is returned from ALESERV DELETE, register 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** ALESERV DELETE has completed successfully.  
**Action:** None. |
| 8                       | **Meaning:** Program error. The caller is not EAX-authorized to the address space specified by the ALET, or the space specified by the ALET is not the primary address space. The entry is not deleted.  
**Action:** Verify that the intended STOKEN is specified. |
| 14                      | **Meaning:** Program or environmental error. The input ALET corresponds to an ALE that is not valid.  
**Action:** Verify that the specified ALET is valid. |
## ALESERV Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 1C                      | **Meaning**: Program error. The caller is holding a lock.  
                          **Action**: Release all locks before calling ALESERV. |
| 20                      | **Meaning**: Program error. The caller is disabled.  
                          **Action**: Enable your program before it issues ALESERV. |
| 24                      | **Meaning**: Program error. AR 1 contained an ALET of 1 on input, or an ALET associated with the caller's PASN-AL.  
                          **Action**: Verify that AR 1 contains either an ALET of 0 or the ALET for the caller's DU-AL. |
| 28                      | **Meaning**: Program error. The caller specified an ALET that is not valid.  
                          **Action**: Verify that the input ALET is valid. |
| 2C                      | **Meaning**: Program error. The caller attempted to delete ALET 0, 1, or 2.  
                          **Action**: Verify that the specified ALET is not ALET 0, 1, or 2. |
| 30                      | **Meaning**: Program error. A problem state caller with PSW key 8 - F tried to delete an entry for a space other than a SCOPE=SINGLE data space.  
                          **Action**: Verify that the ALET supplied represents the intended space. |
| 40                      | **Meaning**: Program or environmental error. The space associated with the input ALET is not valid for cross memory access.  
                          **Action**: None required. However, you may want to take some action based upon your application. |
| 44                      | **Meaning**: Environmental error. The ALE associated with the input ALET represents addressing capability to a deleted or terminated space.  
                          **Action**: None required. However, you may want to discard the specified ALET and possibly take some action based upon your application. |
| 60                      | **Meaning**: System error. An unexpected error occurred. The request was not completed.  
                          **Action**: Retry the request. |
| 78                      | **Meaning**: Program error. A problem state caller with PSW key 8 - F tried to delete an entry from the PASN-AL. The caller is neither the owner nor the creator of the data space, or the PSW key of the caller did not match the storage key of the data space.  
                          **Action**: Change your program's logic so that it does not have to try to delete a data space it did not create or own. |

When control is returned from ALESERV EXTRACT, register 15 contains one of the following hexadecimal return codes:
### ALESERV Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** ALESERV EXTRACT completed successfully. Register 0 contains one of the following reason codes:  
  00 - The access list entry is a public entry.  
  04 - The access list entry is a private entry.  
**Action:** None. |
| 14                      | **Meaning:** Program or environmental error. The input ALET corresponds to an access list entry that is not valid.  
**Action:** Verify that the specified ALET is valid. |
| 24                      | **Meaning:** Program error. AR 1 contained an ALET of 1 on input, or an ALET associated with the caller's PASN-AL.  
**Action:** Verify that AR 1 contains either an ALET of 0 or the ALET for the caller's DU-AL. |
| 28                      | **Meaning:** Program error. The caller specified an ALET that is not valid.  
**Action:** Verify that the input ALET is valid. |
| 3C                      | **Meaning:** Program error. The caller specified an ALET value of 1.  
**Action:** Verify that the specified ALET is other than 1. |
| 40                      | **Meaning:** Program or environmental error. The space associated with the input ALET is not valid for cross memory access.  
**Action:** None required. However, you may want to take some action based upon your application. |
| 44                      | **Meaning:** Environmental error. The access list entry (ALE) associated with the input ALET represents addressing capability to a deleted or terminated space.  
**Action:** None required. However, you may want to discard the specified ALET and possibly take some action based upon your application. |
| 50                      | **Meaning:** Program error. The ALESERV parameter list is not valid.  
**Action:** Verify that your program is not overwriting the parameter list and that the execute form of the macro correctly addresses the parameter list. |
| 58                      | **Meaning:** Program or environmental error. The ALET the caller specified represents an STOKEN for a data space that is no longer accessible.  
**Action:** None required. However, you may want to discard the specified ALET and possibly take some action based upon your application. |
| 60                      | **Meaning:** System error. An unexpected error occurred. The request was not completed.  
**Action:** Retry the request. |

When control is returned from ALESERV SEARCH, register 15 contains one of the following hexadecimal return codes:
### ALESERV Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** ALESERV SEARCH completed successfully. Register 0 contains one of the following hexadecimal reason codes:  
                         | 00 - The access list entry is a public entry.  
                         | 04 - The access list entry is a private entry.  
                         | **Action:** None. |
| 24                      | **Meaning:** Program error. AR 1 contained an ALET of 1 on input or an ALET associated with the caller’s PASN-AL.  
                         | **Action:** Verify that AR 1 contains either an ALET of 0 or the ALET for the caller’s DU-AL. |
| 28                      | **Meaning:** Program error. The caller specified an ALET that is not valid.  
                         | **Action:** Verify that the input ALET is valid. |
| 34                      | **Meaning:** Program error. The caller specified an STOKEN that is not represented on the specified access list.  
                         | **Action:** Verify that the specified STOKEN is on the referenced access list. |
| 48                      | **Meaning:** Program error. The caller specified AL=WORKUNIT but the input ALET indexes into the PASN-AL, or the caller specified AL=PASN but the ALET indexes into the DU-AL.  
                         | **Action:** Change the AL or the ALET parameters to specify the correct AL and ALET combination. |
| 60                      | **Meaning:** System error. An unexpected error occurred. The request was not completed.  
                         | **Action:** Retry the request. |

When control is returned from ALESERV EXTRACTH, register 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** ALESERV EXTRACTH has completed successfully.  
                         | **Action:** None. |
| 24                      | **Meaning:** Program error. AR 1 contained an ALET of 1 on input, or an ALET associated with the caller’s PASN-AL.  
                         | **Action:** Verify that AR 1 contains either an ALET of 0 or the ALET for the caller’s DU-AL. |
| 60                      | **Meaning:** System error. An unexpected error occurred. The request was not completed.  
                         | **Action:** Retry the request. |

### Example of adding an entry to a DU-AL

To add an entry to a DU-AL for a data space, issue the following:
**ALESERV Macro**

ALESERV ADD,STOKEN=DSPCSTKN,ALET=DSPCALET

*DSPCSTKN DS CL8 DATA SPACE STOKEN
DSPCALET DS F DATA SPACE ALET

**ALESERV—List form**

The list form of ALESERV assigns the correct amount of storage for an ALESERV parameter list.

The list form is written as follows:

```
name name
```

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

**Parameters**

The parameters are explained as follows:

- **MF=L**
  - Specifies the list form of the ALESERV macro.
- `,RELATED=any-value`
  - Specifies information used to self document macro by ‘relating’ functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid macro parameter expression.

**ALESERV-Execute form**

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

**Syntax**

The execute form of the macro is written as follows:

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

b

One or more blanks must precede ALESERV.

ALESERV

b

One or more blanks must follow ALESERV.

Valid parameters (required parameters are underlined):

**ADD**
AL, STOKEN, ACCESS, ALET, CHKPT, MF, RELATED

**ADDPASN**
ALET, CHKPT, MF, RELATED

**DELETE**
ALET, MF, RELATED

**EXTRACT**
ALET, CHKPT, MF, RELATED

**SEARCH**
ALET, STOKEN, AL, RELATED, MF

**EXTRACTH**
STOKEN, MF, RELATED

**.ACCESS=PUBLIC**
Default: ACCESS=PUBLIC

**.ACCESS=PRIVATE**

**.AL=WORKUNIT**
Default: AL=WORKUNIT

**.AL=PASN**

**.ALET=alet-addr**
al-addr: RX-type address or register (2) - (12).

**.STOKEN=stoken-addr**
stoken-addr: RX-type address.

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

**.CHKPT=FAIL**
Default: CHKPT=FAIL

**.CHKPT=IGNORE**

**.RELATED=any-value**

Parameters

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

**.MF=(E,cntl addr)**

Specifies the execute form, which uses a remote parameter list. **cntl addr** specifies the address of the remote parameter list, created by a list generated by the list form of the macro.
ALESERV Macro
Chapter 4. ASASYMBM — Substitute text for symbols

Description

**Note**

ASASYMBM is a linkable system service.

Use the ASASYMBM service to substitute text for system symbols. You can explicitly call ASASYMBM to substitute text for system symbols in application or vendor programs. The system calls ASASYMBM automatically for system symbols that are specified in:

- Dynamic allocations
- Job control language (JCL)
- Parmlib members
- System commands.

The caller of ASASYMBM provides an input string to be substituted (a pattern), an output buffer, and optionally a table of system symbols and associated values. ASASYMBM substitutes values for the system symbols that it finds in the input string. ASASYMBM places the results of the substitution in the specified output buffer.

Environment

The requirements for the caller are:

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

Programming requirements

1. To build the parameter area required by ASASYMBM, you must include the ASASYMBP mapping macro (see [z/OS MVS Data Areas](http://www.ibm.com/systems/z/os/zos/bkserv) in z/OS Internet Library at [http://www.ibm.com/systems/z/os/zos/bkserv](http://www.ibm.com/systems/z/os/zos/bkserv)).
2. Before calling ASASYMBM, the caller must provide the following in the ASASYMBP mapping macro:
   - An input string to be substituted (a pattern) and its length
   - An output buffer and its length
   - An area in which to place the return code from ASASYMBM.
   The caller can optionally provide a symbol table and a timestamp.
3. To determine the return code from ASASYMBM, the caller must examine the fullword pointed to by the SYMBPRETURNCODE@ field in the ASASYMBP data area.
4. To determine the length of the output from ASASYMBM, the caller must examine the fullword pointed to by SYMBPTARGETLENGTH@ in the
ASASYMBM service

ASASYMBP mapping macro. The output itself is in the area provided by the caller, which is pointed to by SYMBPTARGET@ in ASASYMBP.

For more information about providing input to ASASYMBM in the ASASYMBP mapping macro, see the section on using the symbol substitution service in [z/OS MVS Programming: Assembler Services Guide](#).

Restrictions

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

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 72-byte save area in the primary address space</td>
</tr>
</tbody>
</table>

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

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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

This service is not appropriate for use in a performance-sensitive area.

Syntax

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

```plaintext
label LINK EP=ASASYMBM, MF=(E, parmarea)
```

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

1. Issue the MVS LOAD macro to load the ASASYMBM service and obtain its entry point address.
2. Issue the CALL macro to call the service. Specify $MF=(E, your\_parmlist)$ on the call.

Parameters

The parameters are explained as follows:

**label**

The name on the macro invocation.

**LINK**

**LINKX**

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

$EP=\text{ASASYMBM}$

Specifies the entry point name for the ASASYMBM service.

$MF=(E,\text{parmarea})$

Specifies the area that you built, mapped by the ASASYMBP macro, that contains the parameter area and optionally points to the system symbol table ASASYMBM is to use.

$SF=(E,\text{parmlist})$

For use with LINKX when your program is reentrant. Before you call LINKX with this parameter, define \textit{parmlist} using the LIST form of LINKX.

Return and reason codes

When the ASASYMBM service returns control to your program, the area pointed to by the SYMBPRETURNCODE@ field of the caller-provided ASASYMBP area contains a return code.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: The ASASYMBM request completed successfully. The system performed the requested substitution. Action: None required.</td>
</tr>
</tbody>
</table>
| 04                      | Meaning: Warning. The caller indicated that the system is to assign a substring of a substitution text to a system symbol. One of the following occurred:  
  • The start position in the substring is either beyond the length of the substitution text or zero.  
  • The length of the substring is either beyond the length of the substitution text or zero.  
  • The length of the substring exceeds the length of the substitution text beyond the specified start position.  
  When the program called ASASYMBM, the SYMBTWARNSUBSTRINGS flag in the ASASYMBP mapping macro indicated that ASASYMBM was to return this return code.  
  The system continues with symbolic substitution.  
  Action: None required. If necessary, see the section on errors in substringing in \textit{z/OS MVS Initialization and Tuning Reference} for information about specifying substrings in system symbols. Ensure that the symbols in the input pattern conform to the rules for substringing. |
### ASASYMBM service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | **Meaning:** Warning. The specified buffer is too small to contain all the substitution text.  
**Action:** Specify a larger target buffer, or continue processing, using the value returned in the fullword pointed to by the SYMBPTARGETLENGTH@ field to determine how much data was placed into the target buffer. |
| 0C                      | **Meaning:** Warning. The length of the text to be substituted in place of a system symbol is null. When the program called ASASYMBM, the SYMBTCHECKNULLSUBTEXT flag in the ASASYMBP mapping macro indicated that ASASYMBM was to return this return code.  
**Action:** None required. |
| 10                      | **Meaning:** Warning. The system did not find any symbols for which it was to substitute text. The substitution process completed normally. When the program called ASASYMBM, the SYMBTWARNNOSUB flag in the ASASYMBP mapping macro indicated that ASASYMBM was to return this return code.  
**Action:** None required. |

### Examples of calls to ASASYMBM

For examples of calls to ASASYMBM, see the section that describes the symbol substitution service in [z/OS MVS Programming: Assembler Services Guide](#).
Chapter 5. ATTACH and ATTACHX — Create a new task

Description

Note: IBM recommends that you use ATTACHX rather than ATTACH.

The ATTACH macro causes the system to create a new task and indicates the entry point in the program to be given control when the new task becomes active. The entry point name that is specified must be a member name or an alias in a directory of a partitioned data set, or must have been specified in an IDENTIFY macro. If the system cannot locate the specified entry point, it abnormally terminates the new subtask.

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

The descriptions of ATTACH and ATTACHX in this book are:

- The standard form of the ATTACH macro, which includes general information about the ATTACH and ATTACHX macros, with some specific information about the ATTACH macro. The syntax of the ATTACH macro is presented, and all ATTACH parameters are explained.
- The standard form of the ATTACHX macro, which includes information specific to the ATTACHX macro and to callers in AR mode.
- The list form of the ATTACH and ATTACHX macros.
- The execute form of the ATTACH and ATTACHX macros.

The new task is a subtask of the originating task; the originating task is the task that was active when you issued the ATTACH macro. The limit and dispatching priorities of the new task are the same as those of the originating task unless modified in the ATTACH macro. The address space control mode (ASC) of the new task is the same as the originating task.

The load module containing the program to be given control is brought into virtual storage if a usable copy is not available in virtual storage. The issuing program can provide an event control block in which termination of the new task is posted and an exit routine to be given control when the new task is terminated.

If you code the ECB or ETXR parameter, you must issue a DETACH macro to remove the subtask from virtual storage before the program that issued the ATTACH macro terminates. If you do not code the ECB or ETXR parameter, the system automatically removes the subtask from virtual storage upon completion of the subtask’s processing. If you specify the ECB parameter in the ATTACH macro, the ECB must be in storage addressable by both the issuer of ATTACH and the system, so that the issuer of ATTACH can wait on it (using the WAIT macro) and the system can post it on behalf of the terminating task.

Environment

The requirements for the caller of ATTACH or ATTACHX are:

Minimum authorization: Problem state and any PSW key
Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
ATTACH and ATTACHX Macros

AMODE: If you use the STAI parameter, 24-bit; otherwise, 24- or 31- or 64-bit.
Note: AMODE 64 is valid only for the ATTACHX macro.

ASC mode: If you use the STAI parameter, primary; otherwise, primary or access register (AR)

Interrupt status: Enabled for I/O and external interrupts

Locks: No locks held

Control parameters: For both primary ASC mode callers and AR ASC mode callers, control parameters must be in the primary address space.

Programming requirements

If your program is in AR mode, issue SYSSTATE ASCENV=AR so the system can generate code that is appropriate for AR mode. If you issue SYSSTATE ASCENV=AR and then issue ATTACH, the system substitutes the ATTACHX macro and issues a message telling you that it made the substitution.

Restrictions

- If the caller is running in 31-bit addressing mode, all input parameters can have addresses greater than 16 megabytes, except for the address of the DCB.
- The caller cannot have an EUT FRR established.
- The parameter list specified for an ESTAI exit must be addressable using a 31-bit address.

Input register information

If you want to pass a parameter list to the new task without coding the PARAM or MF=E parameter, general purpose register (GPR) 1 must contain the address of the list on entry to ATTACH or ATTACHX. Otherwise, before issuing the ATTACH or ATTACHX 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>If GPR 15 contains a return code other than X'00', zero; otherwise, the address of the task control block for the new task</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Zero (the ALET of the task control block address)</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 ATTACH macro is written as follows:

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

b
One or more blanks must precede ATTACH.

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EP=entry name</td>
<td>entry name: Symbol.</td>
</tr>
<tr>
<td>EPLOC=entry name addr</td>
<td>entry name addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>DE=list entry addr</td>
<td>list entry addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,DCB=dcb addr</td>
<td>dcb addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,LPMOD=limit prior nmbr</td>
<td>limit prior nmbr: Symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td>,DPMOD=disp prior nmbr</td>
<td>disp prior nmbr: Symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td>,PARAM=(addr)</td>
<td>addr: A-type address</td>
</tr>
<tr>
<td>,PARAM=(addr),VL=1</td>
<td>Note: addr is one or more addresses, separated by commas. For example, PARAM=(addr,addr,addr)</td>
</tr>
<tr>
<td>,ECB=ecb addr</td>
<td>ecb addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,ETXR=exit rtn addr</td>
<td>exit rtn addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,GSPV=subpool nmbr</td>
<td>subpool nmbr: Symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td>,GSPL=subpool list addr</td>
<td>subpool list addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,SHSPV=subpool nmbr</td>
<td>subpool nmbr: Symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td>,SHSPL=subpool list addr</td>
<td>subpool list addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,SZERO=YES</td>
<td>Default: SZERO=YES</td>
</tr>
<tr>
<td>,SZERO=NO</td>
<td></td>
</tr>
<tr>
<td>,TASKLIB=dcb addr</td>
<td>dcb addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,STAI=(exit addr)</td>
<td>exit addr: A-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,STAI=(exit addr,parm addr)</td>
<td>parm addr: A-type address, or register (2) - (12).</td>
</tr>
</tbody>
</table>
**ATTACH and ATTACHX Macros**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>,ESTAI=(exit addr)</td>
<td>Note: AR mode callers and 31-bit callers must not use STAI.</td>
</tr>
<tr>
<td>,ESTAI=(exit addr, parm addr)</td>
<td></td>
</tr>
<tr>
<td>,PURGE=QUIESCE</td>
<td>Note: PURGE may be specified only if STAI or ESTAI is specified. Default for STAI: PURGE=QUIESCE Default for ESTAI: PURGE=NONE</td>
</tr>
<tr>
<td>,PURGE=NONE</td>
<td></td>
</tr>
<tr>
<td>,PURGE=HALT</td>
<td></td>
</tr>
<tr>
<td>,ASYNCH=NO</td>
<td>Default for STAI: ASYNCH=NO Default for ESTAI: ASYNCH=NO Note: ASYNCH may be specified only if STAI or ESTAI is specified.</td>
</tr>
<tr>
<td>,ASYNCH=YES</td>
<td></td>
</tr>
<tr>
<td>,TERM=NO</td>
<td>Note: TERM may be specified only if ESTAI is specified. Default: TERM=NO</td>
</tr>
<tr>
<td>,TERM=YES</td>
<td></td>
</tr>
<tr>
<td>,ALCOPY=NO</td>
<td>Default: ALCOPY=NO</td>
</tr>
<tr>
<td>,ALCOPY=YES</td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
</tbody>
</table>

**Parameters**

The parameters are explained as follows:

- **EP=entry name**
- **EPLOC=entry name addr**
- **DE=list entry addr**

Specifies the entry name, the address of the entry name, or the address of the name field of a 62-byte list entry for the entry name that was constructed using the BLDL macro. If EPLOC is coded, the name must be padded to eight bytes, if necessary.

When you use the DE parameter with the ATTACH macro, DE specifies the address of a list that was created by a BLDL macro. You must issue the BLDL and the ATTACH from the same task; otherwise, the system abnormally terminates the program with a completion code of X’106’. **Do not issue an ATTACH or a DETACH between issuances of the BLDL and ATTACH.**

The system ignores the information you specify on the DE parameter if the parameter does one of the following:

- Specifies an entry in an authorized library (that is, defined in IEAAPFx member of SYS1.PARMLIB)
- Requests access to a program or library that is controlled by the system authorization facility (SAF).

Instead, the system uses the BLDL macro to construct a new list entry containing the DE information.

The contents of the GPRs on entry to the subtask are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Address of the user parameter list if specified on either the PARAM or MF=E parameters; otherwise, contains whatever GPR 1 contained at the time the ATTACH macro was issued.</td>
</tr>
<tr>
<td>2-12</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>
ATTACH and ATTACHX Macros

13 Address of a standard save area.
14 Return address. Bit 0 is 0 if the subtask routine gets control in 24-bit addressing mode; bit 0 is 1 if the subtask routine gets control in 31-bit addressing mode.
15 When the subtask routine is to run in 24-bit or 31-bit addressing mode, the entry point address of the subtask routine.

When the subtask routine is to run in 64-bit addressing mode, it is expected to use relative branching and register 15 contains a value that can be used to determine the addressing mode of the issuer of the ATTACH or ATTACHX macro as follows:

Issuer AMODE 24: X'FFFFF000'
Issuer AMODE 31: X'FFFFF002'
Issuer AMODE 64: X'FFFFF004'

The contents of the ARs on entry to the subtask are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Zero if you specified a user parameter list on either the PARAM or MF=E parameters; otherwise, contains whatever AR 1 contained at the time the ATTACH macro was issued.</td>
</tr>
<tr>
<td>2-12</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

,DCB=dcb addr
Specifies the address of the data control block for the partitioned data set containing the entry name.

Note: The DCB must be opened before the ATTACH macro is issued and must be the DCB used in the BLDL that built the 62-byte DE list entry. The DCB must remain open until the subtask becomes active, and it should not be closed immediately following the ATTACH.

Note: DCB must reside in 24-bit addressable storage.

,LPMOD=limit prior nmbr
Specifies the number (0 to 255) to be subtracted from the current limit priority of the originating task. The resulting number is the limit priority of the new task, with a higher number representing a higher limit priority.

If you omit this parameter, the current limit priority of the originating task is assigned as the limit priority of the new task.

,DPMOD=disp prior nmbr
Specifies the signed number (−255 to +255) to be algebraically added to the current dispatching priority of the originating task. The resulting number is assigned as the dispatching priority of the new task, with a higher number representing a higher dispatching priority. If, however, the resulting number is higher than the limit priority of the new task, the limit priority is assigned as the dispatching priority.

If a register is designated, a negative number must be in two's complement form in the register. If you omit this parameter, the dispatching priority assigned is the smaller of either the new task's limit priority or the originating task's dispatching priority.

,PARAM=(addr)
,PARAM=(addr),VL=1
Specifies an address or addresses to be passed to the attached program.
ATTACH and ATTACHX Macros

ATTACH expands each address inline to a fullword on a fullword boundary, in the order designated, building a parameter list. When the program receives control, register 1 contains the address of the first word of the parameter list.

Specify VL=1 only if the called program can be passed a variable number of parameters. VL=1 causes the high-order bit of the last address to be set to 1; the bit can be checked to find the end of the list.

\texttt{,ECB=ecb addr}

Specifies the address of an event control block (ECB) for the new task that the system will use to indicate when the new task terminates. The ECB must be in storage so that the issuer of ATTACH can wait on it (using the WAIT macro) and the system can post it on behalf of the terminating task. The return code (if the task is terminated normally) or the completion code (if the task is terminated abnormally) is also placed in the event control block. If you code this parameter, you must issue a DETACH macro to remove the subtask from virtual storage after the subtask terminates. The system assumes that the ECB is in the home address space.

\texttt{,ETXR=exit rtn addr}

Specifies the address of the end-of-task exit routine to be given control after the new task is normally or abnormally terminated. The exit routine receives control when the originating task becomes active after the subtask is terminated, and must be in virtual storage when required. If you code this parameter, you must issue a DETACH macro to remove the subtask from the system after the subtask terminates.

The exit routine runs asynchronously under the originating task. The routine receives control in the addressing mode of the issuer of the ATTACH macro. The system abnormally ends a task with completion code X'72A' if the task attempts to create two subtasks with the same exit routine in different addressing modes. Upon entry, the routine has an empty dispatchable unit access list (DU-AL). To establish addressability to a data space created by the originating task and shared with the terminating subtask, the routine can issue the ALESERV macro with the ADD parameter, and specify the STOKEN of the data space.

The exit routine receives control with the following environment:

- **Authorization:** Problem state, PSW key is the same as TCB key of the issuer of the ATTACH macro.
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** Same as the issuer of the ATTACH macro
- **ASC mode:** Primary
- **Interrupt status:** Enabled for I/O and external interrupts
- **Locks:** No locks held
- **Control parameters:** Not applicable.

The contents of the GPRs when the exit routine receives control are as follows:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Address of the task control block for the task that was terminated.</td>
</tr>
<tr>
<td>2-12</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>13</td>
<td>Address of a save area provided by the system.</td>
</tr>
<tr>
<td>14</td>
<td>Return address (to the system).</td>
</tr>
<tr>
<td>15</td>
<td>Address of the exit routine.</td>
</tr>
</tbody>
</table>
ATTACH and ATTACHX Macros

The contents of the ARs when the exit routine receives control are:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>Zero</td>
</tr>
<tr>
<td>2-12</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>13-15</td>
<td>Zeros</td>
</tr>
</tbody>
</table>

The exit routine is responsible for saving and restoring the registers.

,.GSPV=subpool nmbr  
,.GSPL=subpool list addr

Specifies a virtual storage subpool number less than 128 or the address of a list of virtual storage subpool numbers each less than 128. Except for subpool zero, ownership of each of the specified subpools is assigned to the new task. Although it can be specified, subpool zero cannot be transferred. When ownership of a subpool is transferred, programs of the originating task can no longer obtain or release the associated virtual storage areas.

If GSPL is specified, the first byte of the list contains the number of following bytes in the list; each of the following bytes contains a virtual storage subpool number.

,.SHSPV=subpool nmbr  
,.SHSPL=subpool list addr

Specifies a virtual storage subpool number less than 128 or the address of a list of virtual storage subpool numbers each less than 128. Programs of both originating task and the new task can use the associated virtual storage areas.

If SHSPL is specified, the first byte of the list contains the number of remaining bytes in the list; each of the following bytes contains a virtual storage subpool number.

,.SZERO=YES  
,.SZERO=NO

Specifies whether subpool 0 is to be shared with the subtask. YES specifies that subpool 0 is to be shared; NO specifies that subpool 0 is not to be shared.

,.TASKLIB=dcb addr

Specifies the address of the DCB for the library to be used as the attached task's library. Otherwise, the task library is propagated from the originating task. (Note: The DCB must be opened before the ATTACH macro is executed.) SYS1.LINKLIB is the last library searched. If the DCB address specifies SYS1.LINKLIB, the search begins with SYS1.LINKLIB, goes through other libraries, and ends with SYS1.LINKLIB. The system abnormally terminates the attached task with a completion code of X'806' if the requested module is not in the task library and is not in the other libraries searched.

See “Location of the Load Module” in z/OS MVS Programming: Assembler Services Guide for additional information on using the TASKLIB parameter.

Note: DCB must reside in 24-bit addressable storage.

,.STAI=(exit addr)  
,.STAI=(exit addr,parm addr)  
,.ESTAI=(exit addr)  
,.ESTAI=(exit addr,parm addr)

Specifies whether a STAI or ESTAI recovery routine is to be defined; any recovery routines defined for the originating task are propagated to the new task.
ATTACH and ATTACHX Macros

The *exit addr* specifies the address of the STAI or ESTAI recovery routine that is to receive control if the subtask encounters an error; the recovery routine must be in virtual storage at the time of the error. The *parm addr* is the address of a parameter list which may be used by the STAI or ESTAI recovery routine. The address must be 24-bit for STAI and 31-bit for ESTAI.

ATTACHX processing passes control to an ESTAI recovery routine in the addressing mode of the issuer of the ATTACHX macro. A STAI exit routine can run only in 24-bit addressing mode. If a caller in the wrong addressing mode or AR mode specifies the STAI parameter on the ATTACH macro, the caller ends abnormally with a completion code of X'52A'.

**PURGE**<br>Specifies what action is to be taken with regard to I/O operations if the subtask encounters an error. No action may be specified (NONE), a halting of I/O operations may be requested (HALT), or a quiescing of I/O operations may be indicated (QUIESCE).

**Note:** You need to understand PURGE processing before using this parameter. For information about PURGE processing, see [z/OS DFSMSdfp Advanced Services](z/OS DFSMSdfp Advanced Services).

**ASYNCH**<br>Specifies whether asynchronous exits are to be allowed when a subtask encounters an error.

ASYNCH=YES must be coded if:
- Any supervisor services that require asynchronous interruptions to complete their normal processing are going to be requested by the recovery routine.
- PURGE=QUIESCE is specified for any access method that requires asynchronous interruptions to complete normal input/output processing.
- PURGE=NONE is specified and the CHECK macro is issued in the recovery routine for any access method that requires asynchronous interruptions to complete normal input/output processing.

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

**TERM**<br>Specifies whether the recovery routine associated with the ESTAI request is also to be scheduled in the following situations:
- System-initiated logoff
- Job step timer expiration
- Wait time limit for job step exceeded
- DETACH macro without the STAE=YES parameter issued from a higher-level task (possibly by the system if the higher-level task encountered an error)
- Operator cancel
- Error on a higher-level task
- Error in the job step task when a non-job-step task issued the ABEND macro with the STEP parameter
ATTACH and ATTACHX Macros

- z/OS UNIX System Services is canceled and the user's task is in a wait in the z/OS UNIX System Services kernel.

\texttt{,ALCOPY=NO}

\texttt{,ALCOPY=YES}

Determines the contents of the new task's access list and determines the extended authorization index (EAX) value for the new task. ALCOPY=NO gives the new task an EAX of zero and a null access list. ALCOPY=YES gives the new task:

- The same extended authorization index (EAX) as the caller
- A copy of the caller's DU-AL.

The default is ALCOPY=NO.

\texttt{,RELATED=value}

Specifies information used to self-document a macro by "relating" functions or services to corresponding functions or services. The format and contents of the information specified are at the discretion of the user, and may be any valid coding values.

ABEND codes

The caller of ATTACH or ATTACHX might receive one of the following ABEND codes:

<table>
<thead>
<tr>
<th>ABEND Code</th>
<th>Associated Reason Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>12A</td>
<td>0, 4</td>
</tr>
<tr>
<td>22A</td>
<td>0</td>
</tr>
<tr>
<td>42A</td>
<td>None</td>
</tr>
<tr>
<td>52A</td>
<td>0, 4, 8</td>
</tr>
<tr>
<td>72A</td>
<td>0, 4, 8, C, 10, 14</td>
</tr>
<tr>
<td>82A</td>
<td>None</td>
</tr>
<tr>
<td>92A</td>
<td>0, 4, 8, C, 10, 14, 18</td>
</tr>
</tbody>
</table>

Note: ABEND code 92A results from an error not directly caused by the caller.

Return and reason codes

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | Meaning: Successful completion.  
Action: None. |
| 04                      | Meaning: Program error. ATTACH was issued in a STAE exit; processing not completed.  
Action: Change your program so that the ATTACH is not issued by a STAE exit routine. |
| 08                      | Meaning: Environmental error. Insufficient storage available for control block for STAI/ESTAI request; processing not completed.  
Action: Retry the request. |
ATTACH and ATTACHX Macros

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C                      | **Meaning:** Incorrect exit routine address or incorrect parameter list address specified with STAI parameter; processing not completed.  
**Action:** Ensure that the exit routine and parameter list address are correct. |

**Note:** It is possible for the originating task to obtain return code 00, and still not have the subtask successfully created (for example, if the entry name could not be found). In such cases, the new subtask is abnormally terminated.

**Example 1**
Cause the program named in the list to be attached. Establish RTN as an end-of-task exit routine.

```
ATTACH DE=LISTNAME,ETXR=RTN
```

**Example 2**
Cause PROGRAM1 to be attached, share subpool 5, wait on WORD1 to synchronize processing with that of the subtask, and establish EXIT1 as an ESTAI exit.

```
ATTACH EP=PROGRAM1,SHSPV=5,ECB=WORD1,ESTAI=(EXIT1)
```

**Example 3**
Cause PROGRAM1 to be attached and share subpool zero. The subtask is to receive control:
- With the same extended authorization index (EAX) as the caller.
- With a copy of the caller's DU-AL.

```
TESTCASE CSECT
   ATTACH EP=PROGRAM1,SZERO=YES,ALCOPY=YES
   END TESTCASE
```

**Example 4**
Usage of the SF and MF parameters.

```
MVC ATTACH_EXEC,ATTACH_LIST Copy static plist to dynamic
*
   ATTACHX
      PARAM=(PARM1,PARM2,PARM3),
      MF=(E,REMOTE_PLIST),
      SF=(E,ATTACH_EXEC)
* (in the module's static area)
*
   ATTACH_LIST ATTACHX EP=PROGRAM1,SZERO=YES,ALCOPY=YES,SF=L
*
   (in the module's dynamic area)

REMOTE_PLIST DS 3F
```
ATTACHX—Create a new task

The ATTACHX macro creates a new task and indicates the entry point in the program to be given control when the new task becomes active. The ASC mode of the new task is the same as the ASC mode of the issuer of ATTACHX.

At entry to the attached task, if the caller specifies a user parameter list on the PARAM parameter or by issuing the execute form of the macro with MF=E:

- GPR 1 contains the address of the user parameter list.
- If the caller of the ATTACHX macro is in AR mode, AR 1 contains an ALET of 0.

All parameters that are valid for ATTACH are also valid for ATTACHX.

Syntax

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

```plaintext
name name

b

ATTACHX

b
```

EP=entry name

EPLOC=entry name addr

DE=list entry addr

,DCB=dcb addr

,LPMOD=limit prior nmbr

,DPMOD=disp prior nmbr

,PARAM=(addr)

,PARAM=(addr),VL=1

,PLIST4=YES

,PLIST4=NO

,PLIST8=YES

,PLIST8=NO

,PLIST8ARALETS=NO

,PLIST8ARALETS=YES

Note: `addr` is one or more addresses, separated by commas. For example, `PARAM=(addr,addr,addr)`

Default: None.
ATTACH and ATTACHX Macros

,.ECB=ecb addr  
  ECB addr: A-type address, or register (2) - (12).

,.ETXR=exit rtn addr  
  Exit rtn addr: A-type address, or register (2) - (12).

,.GSPV=subpool nmbr
,.GSPL=subpool list addr  
  Subpool nmbr: Symbol, decimal digit, or register (2) - (12).
  Subpool list addr: A-type address, or register (2) - (12).

,.SHSPV=subpool nmbr
,.SHSPL=subpool list addr  
  Subpool nmbr: Symbol, decimal digit, or register (2) - (12).
  Subpool list addr: A-type address, or register (2) - (12).

,.SZERO=YES
,.SZERO=NO  
  Default: SZERO=YES

,.TASKLIB=dcb addr  
  Dcb addr: A-type address, or register (2) - (12).

,.STAI=(exit addr)
,.STAI=(exit addr,parm addr)  
  Exit addr: A-type address, or register (2) - (12).
  Parm addr: A-type address, or register (2) - (12).
  Note: AR mode callers and 31-bit callers must not use STA.

,.ESTAI=(exit addr)  
  Note: SDWALOC31 may be specified only when ESTAI is specified.
  Default: SDWALOC31=NO

,.ESTAI=(exit addr,parm addr)  
  Note: Specify PURGE only if you specify ESTAI.
  Default for ESTAI: PURGE=NONE

,.PURGE=QUIESCE
,.PURGE=NONE
,.PURGE=HALT  
  Note: Specify ASYNCH only if you specify ESTAI.
  Default for ESTAI: ASYNCH=YES

,.ASYNCH=NO
,.ASYNCH=YES  
  Note: Specify TERM only if you specify ESTAI.
  Default: TERM=NO

,.TERM=NO
,.TERM=YES  
  Note: Specified ALCOPY=NO

,.ALCOPY=NO
,.ALCOPY=YES  
  Default: ALCOPY=NO

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

,.KEY=PROP
,.KEY=NINE  
  Default: KEY=PROP

,.PKM=SYSTEM_RULES
,.PKM=REPLACE  
  Default: PKM=SYSTEM_RULES

Parameters

The parameters are as explained under ATTACH, with the following exceptions:

,.PARAM=(addr)  
  Specifies an address or addresses to be passed to the attached task.

ATTACHX expands each address inline to a fullword boundary and builds a
ATTACH and ATTACHX Macros

parameter list with the addresses in the order specified. When the attached task receives control, register 1 contains the address of the parameter list. When PARAM is not specified, ATTACHX passes GPR1 and AR1 unchanged to the attached routine.

When an AR mode caller uses either:

- a parameter list with 4 bytes per entry; or
- a parameter list with 8 bytes per entry and specifies PLIST8ARALETS=YES,

the addresses passed to the subtask are in the first part of the parameter list and their associated ALETs are in the second part. For a non-AR mode caller, or for an AR mode caller using a parameter list with 8 bytes per entry without PLIST8ARALETS=YES, ALETs are not passed in the parameter list. When ALETs are passed in the parameter list, the ALETs occupy consecutive 4-byte fields, whether the parameter list is 4 or 8 bytes per entry. See the description of the PLIST4 and PLIST8 keywords below for more information about controlling the bytes-per-entry in the parameter list. See the description of the PLIST8ARALETS keyword below for more information about ALETs and 8-bytes-per-entry parameter lists. See "User parameters" on page 4 for an example of passing a parameter list in AR mode.

When using a 4-bytes-per-entry parameter list, specify VL=1 when you pass a variable number of parameters. VL=1 results in setting the high-order bit of the last address to 1. The 1 in the high-order bit identifies the last address parameter (which is not the last word in the list when the ALETs are also saved). When using an 8-bytes-per-entry parameter list, VL=1 is not valid.

Note: If you specify only one address for PARAM= and you are not using register notation, you do not need to enter the parentheses.

,PLIST4=YES
,PLIST4=NO

,PLIST8=YES
,PLIST8=NO

Defines the size of the parameter list entries for a parameter list to be built by ATTACHX based on the PARAM keyword.

PLIST4 and PLIST8 cannot be specified together. If neither is specified, the default is:

- If running AMODE 64, PLIST8=YES
- If not running AMODE 64, PLIST4=YES

If running AMODE 64 and PLIST4=YES is specified, the system builds a 4-bytes-per-entry parameter list just as it would if the program were running AMODE 24 or AMODE 31 and did not specify PLIST4 or PLIST8.

If running AMODE 24 or AMODE 31 and PLIST8 is specified, the system builds an 8-bytes-per-entry parameter list just as it would if the program were running AMODE 64 and did not specify PLIST4 or PLIST8.

,PLIST8ARALETS=NO
,PLIST8ARALETS=YES

If there is to be an 8-byte-per-entry parameter list and the invoker is in AR mode, indicates if the parameter list is also to contain the ALETs associated with the addresses. Otherwise, this parameter is ignored.
ATTACH and ATTACHX Macros

**,PLIST8ARALETs=NO**
Indicates that the 8-byte-per-entry parameter list is to consist of just the 8-byte addresses.

**,PLIST8ARALETs=YES**
Indicates that the 8-byte-per-entry parameter list is to consist of the following two parts:
- All the 8-byte addresses,
- All the associated ALETs in consecutive 4-byte fields.

**,SDWALOC31=NO**
**,SDWALOC31=YES**
Specifies the location of the ESTAI's SDWA.
If using ESTAI and SDWALOC31=YES, then the SDWA is in 31–bit storage.
If using ESTAI and SDWALOC31=NO, then the SDWA is in 24–bit storage.

**,KEY=PROP**
**,KEY=NINE**
PROP specifies that the protection key of the newly created task should be propagated from the task using ATTACH. NINE specifies that the protection key of the newly created task should be nine.

You can use KEY=NINE to help to prevent the attached task from inadvertently modifying storage owned by the attaching task, since a program running in with PSW key 9 cannot modify storage in any other PSW key. The following parameters are not valid when KEY=NINE is specified: GSPPL, GSPV, SHSPL, and SHSVP. In addition, if you specify KEY=NINE, you must specify SZERO=NO.

Within a task that was attached with the KEY=NINE parameter:
- the system-provided save area is above 16M (for a non-KEY=NINE task, the save area is below 16M)
- a re-entrant program, whether from an APF-authorized concatenation or not, is placed into key 0 storage (for a non-KEY=NINE task, only re-entrant programs from an APF-authorized concatenation are placed into key 0 storage).

**,PKM=SYSTEM_RULES**
**,PKM=REPLACE**
SYSTEM_RULES specifies that the system should determine the appropriate PSW key mask using the following rules:
- If KEY=ZERO, the PSW key mask represents key 0 plus key 9.
- If KEY=PROP, but the mother task's initial key does not match the mother task's current key, the PSW key mask represents the PSW key of the daughter task plus key 9.
- If KEY=PROP and the mother task's initial key matches the mother task's current key, or if KEY=NINE, the PSW key mask represents the mother task's initial key plus the mother task's initial PSW key mask plus the PSW key of the daughter task plus key 9.

REPLACE specifies that the PSW key mask is to be replaced with a value representing the PSW key of the daughter task plus key 9.

The default is PKM=SYSTEM_RULES.
ATTACH and ATTACHX Macros

Example

With the caller in AR ASC mode, cause PROGRAM1 to be attached and share subpool zero. The subtask is to receive control:
- With the same extended authorization index (EAX) as the caller.
- With a copy of the caller’s DU-AL.
- Executing in AR ASC Mode.

```
TESTCASE CSECT
    SYSSTATE ASCENV=AR
    ATTACHX EP=PROGRAM1,SZERO=YES,ALCOPY=YES
END TESTCASE
```

ATTACH and ATTACHX—List form

Two parameter lists are used on ATTACH or ATTACHX: a control parameter list and an optional user parameter list. You can construct only the control parameter list in the list form. Address parameters to be passed in a parameter list to the attached task can be provided using the list form of the CALL macro. This parameter list can be referred to in the execute form.

Syntax

The list form of the ATTACH and ATTACHX is written as follows:

```
name name
/bslash
ATTACH
ATTACHX
/bslash
name

EP=entry name
EPLOC=entry name addr
DE=list entry addr
,DCB=dcb addr
,LPMOD=limit prior nmbr
,DPMOD=disp prior nmbr
,PLIST8ARALETs=NO
,PLIST8ARALETs=YES
,ECB=ecb addr
,ETXR=exit rtn addr
```

- `name`: Symbol. Begin `name` in column 1.
- `b`: One or more blanks must precede ATTACH or ATTACHX.
- `ATTACH`, `ATTACHX`: One or more blanks must follow ATTACH or ATTACHX.
- `EP=entry name`: Symbol.
- `EPLOC=entry name addr`: `entry name addr`: A-type address.
- `DE=list entry addr`: `list entry addr`: A-type address.
- `,DCB=dcb addr`: `dcb addr`: A-type address.
- `,LPMOD=limit prior nmbr`: `limit prior nmbr`: Symbol or decimal digit.
- `,DPMOD=disp prior nmbr`: `disp prior nmbr`: Symbol or decimal digit.
- `,PLIST8ARALETs=NO`: Default: PLIST8ARALETs=NO
- `Note`: PLIST8ARALETs is valid only with ATTACHX.
- `,ECB=ecb addr`: `ecb addr`: A-type address.
- `,ETXR=exit rtn addr`: `exit rtn addr`: A-type address.
### ATTACH and ATTACHX Macros

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GSPV=</td>
<td>subpool nmbr: Symbol or decimal digit.</td>
</tr>
<tr>
<td>GSPL=</td>
<td>subpool list addr: A-type address.</td>
</tr>
<tr>
<td>SHSPV=</td>
<td>subpool nmbr: Symbol or decimal digit.</td>
</tr>
<tr>
<td>SHSPL=</td>
<td>subpool list addr: A-type address.</td>
</tr>
<tr>
<td>SZERO=YES</td>
<td>Default: SZERO=YES</td>
</tr>
<tr>
<td>TASKLIB=</td>
<td>dcb addr: A-type address.</td>
</tr>
<tr>
<td>STAI=(exit addr)</td>
<td>exit addr: A-type address.</td>
</tr>
<tr>
<td>STAI=(exit addr,parm addr)</td>
<td>parm addr: A-type address.</td>
</tr>
<tr>
<td>ESTAI=(exit addr)</td>
<td>Note: AR mode callers and 31-bit callers must not use STA.</td>
</tr>
<tr>
<td>ESTAI=(exit addr,parm addr)</td>
<td>Note: SDWALOC31 is valid only with ATTACHX AND when ESTAI is specified.</td>
</tr>
<tr>
<td>SDWALOC31=NO</td>
<td>Default: SDWALOC31=NO</td>
</tr>
<tr>
<td>PURGE=QUIESCE</td>
<td>Note: PURGE may be specified only if STAI or ESTAI is specified.</td>
</tr>
<tr>
<td>PURGE=NONE</td>
<td>Default for STAI: PURGE=QUIESCE</td>
</tr>
<tr>
<td>PURGE=HALT</td>
<td>Default for ESTAI: PURGE=NONE</td>
</tr>
<tr>
<td>ASYNCH=NO</td>
<td>Note: ASYNCH may be specified only if STAI or ESTAI is specified.</td>
</tr>
<tr>
<td>ASYNCH=YES</td>
<td>Default for STAI: ASYNCH=NO</td>
</tr>
<tr>
<td>TERM=NO</td>
<td>Default for ESTAI: ASYNCH=YES</td>
</tr>
<tr>
<td>TERM=YES</td>
<td>Note: TERM may be specified only if ESTAI is specified.</td>
</tr>
<tr>
<td>ALCOPY=NO</td>
<td>Default: ALCOPY=NO</td>
</tr>
<tr>
<td>ALCOPY=YES</td>
<td>Note: ALCOPY=NO</td>
</tr>
<tr>
<td>RELATED=value</td>
<td>value: Any valid macro keyword specification.</td>
</tr>
<tr>
<td>KEY=PROP</td>
<td>Default: KEY=PROP</td>
</tr>
<tr>
<td>KEY=NINE</td>
<td>Note: KEY=NINE is valid only when using ATTACHX.</td>
</tr>
<tr>
<td>PKM=SYSTEM_RULES</td>
<td>Default: PKM=SYSTEM_RULES</td>
</tr>
<tr>
<td>PKM=REPLACE</td>
<td>Note: PKM is valid only when using ATTACHX.</td>
</tr>
<tr>
<td>SF=L</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

Some parameters in the syntax diagram are only available on the ATTACHX macro. If you are using the ATTACH macro, check the standard form to ensure that the parameters that you want to use are supported by that macro.

The parameters are explained under the standard form of the ATTACH or ATTACHX macro, with the following exception:
Specifications for the ATTACH and ATTACHX macros.

**ATTACH and ATTACHX—Execute form**

Two parameter lists are used on ATTACH and ATTACHX; a control parameter list and an optional user parameter list to be passed to the attached task. Either or both of these parameter lists can be remote (that is, in an area you specifically obtained); you can use the execute form of ATTACH and ATTACHX to refer to or modify them. If only the user parameter list is remote, parameters that require use of the control parameter list cause that list to be constructed inline as part of the macro expansion.

For programs in AR mode, ATTACHX builds the parameter list so that the addresses passed to the system are in the first half of the parameter list and their corresponding ALETs are in the last half of the list. Therefore, the parameter list for callers in AR mode is twice as long as the parameter list for callers in primary mode for the same number of addresses.

**Syntax**

The execute form of the ATTACH and ATTACHX is written as follows:

```plaintext
| name | name: Symbol. Begin name in column 1. |
| b    | One or more blanks must precede ATTACH or ATTACHX. |

ATTACH
ATTACHX
b
One or more blanks must follow ATTACH or ATTACHX.

| EP=entry name | entry name: Symbol. |
| EPLOC=entry addr | entry name addr: RX-type address, or register (2) - (12). |
| DE=list entry addr | list entry addr: RX-type address, or register (2) - (12). |
| ,DCB=dcb addr | dcb addr: RX-type address, or register (2) - (12). |
| ,LPMOD=limit prior nmbr | limit prior nmbr: Symbol, decimal digit, or register (2) - (12). |
| ,DPMOD=disp prior nmbr | disp prior nmbr: Symbol, decimal digit, or register (2) - (12). |
| ,PARAM=(addr) | addr: RX-type address |
| ,PARAM=(addr),VL=1 | Note: addr is one or more addresses, separated by commas. For example, PARAM=(addr,addr,addr) |
| ,PLIST4=YES | PLIST4 is valid only with ATTACHX. |
| ,PLIST4=NO | Default: None. |
| ,PLIST8=YES | PLIST8 is valid only with ATTACHX. |
| ,PLIST8=NO | Default: None. |
```
**ATTACH and ATTACHX Macros**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Default</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>,PLIST8ARALETs</td>
<td>PLIST8ARALETs=NO</td>
<td>PLIST8ARALETs is valid only with ATTACHX.</td>
</tr>
<tr>
<td>,ECB=ecb addr</td>
<td></td>
<td>ecb addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,ETXR=exit rtn addr</td>
<td></td>
<td>exit rtn addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,GSPV=subpool nmbr</td>
<td></td>
<td>subpool nmbr: Symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td>,SHSPV=subpool nmbr</td>
<td></td>
<td>subpool nmbr: Symbol, decimal digit, or register (2) - (12).</td>
</tr>
<tr>
<td>,SZERO=YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>,SZERO=NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>,TASKLIB=dcb addr</td>
<td></td>
<td>dcb addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,STAI=(exit addr)</td>
<td></td>
<td>exit addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,ESTAI=(exit addr)</td>
<td></td>
<td>exit addr: RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>SDWALOC31=YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SDWALOC31=NO</td>
<td></td>
<td>SDWALOC31 is valid only when using ATTACHX AND when ESTAI is specified.</td>
</tr>
<tr>
<td>,PURGE=QUIESCE</td>
<td></td>
<td>PURGE may be specified only if STAI or ESTAI is specified.</td>
</tr>
<tr>
<td>,PURGE=NONE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>,PURGE=HALT</td>
<td></td>
<td></td>
</tr>
<tr>
<td>,ASYNCH=NO</td>
<td></td>
<td>ASYNCH may be specified only if STAI or ESTAI is specified.</td>
</tr>
<tr>
<td>,ASYNCH=YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>,TERM=NO</td>
<td></td>
<td>TERM may be specified only if ESTAI is specified.</td>
</tr>
<tr>
<td>,TERM=YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>,ALCOPY=NO</td>
<td></td>
<td>ALCOPY=NO</td>
</tr>
<tr>
<td>,ALCOPY=YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>,RELATED=value</td>
<td></td>
<td>value: Any valid macro keyword specification.</td>
</tr>
<tr>
<td>,KEY=PROP</td>
<td></td>
<td>KEY=PROP</td>
</tr>
<tr>
<td>,KEY=NINE</td>
<td></td>
<td>KEY=NINE is valid only when using ATTACHX</td>
</tr>
<tr>
<td>,PKM=SYSTEM_RULES</td>
<td></td>
<td>PKM=SYSTEM_RULES</td>
</tr>
<tr>
<td>,PKM=REPLACE</td>
<td></td>
<td>PKM is valid only when using ATTACHX.</td>
</tr>
<tr>
<td>,MF=(E, prob addr)</td>
<td></td>
<td>prob addr: RX-type address, or register (1) or (2) - (12).</td>
</tr>
<tr>
<td>,SF=(E, ctrl addr)</td>
<td></td>
<td>ctrl addr: RX-type address, or register (2) - (12) or (15).</td>
</tr>
</tbody>
</table>
Parameters

Some parameters in the syntax diagram are only available on the ATTACHX macro. If you are using the ATTACH macro, check the standard form to ensure that the parameters that you want to use are supported by that macro.

The parameters are explained under the standard form of the ATTACH or ATTACHX, with the following exceptions:

\[ MF=(E,prob\ addr) \]
\[ SF=(E,ctrl\ addr) \]
\[ MF=(E,prob\ addr), SF=(E,ctrl\ addr) \]

Specifies the execute form of ATTACH or ATTACHX using either a remote user parameter list or a remote control parameter list.

For a caller in AR mode who specifies MF=E, the parameter list that ATTACH or ATTACHX generates for the PARAM parameter is twice as long as the parameter list generated for primary mode callers.

Notes:

1. If STAI is specified on the execute form, the following fields are overlaid in the control parameter list: exit addr, parm addr, PURGE, and ASYNCH. If parm addr is not specified, zero is used; if PURGE or ASYNCH are not specified, defaults are used.

2. If ESTAI is specified on the execute form, then the following fields are overlaid in the control parameter list: exit addr, parm addr, PURGE, ASYNCH, and TERM. If parm addr is not specified, zero is used; if PURGE, ASYNCH, or TERM are not specified, defaults are used.

3. If the STAI or ESTAI is to be specified, it must be completely specified on either the list or execute form, but not on both forms.

4. If SZERO is not specified on the list or execute form, the default is SZERO=YES. If SZERO=NO is specified on either the list form or a previous execute form using the same SF=L, then SZERO=YES is ignored for any following execute forms of the macro. Once SZERO=NO is specified, it is in effect for all users of that list.
ATTACH and ATTACHX Macros
Chapter 6. BLDMPB — Build a message parameter block

Description

The BLDMPB macro builds the fixed portion of a message parameter block (MPB). If you are writing a new application or adding new messages to an existing application, you can place the message text in the install message files rather than in the application code. To translate message text that exists only in the install message files, you need to build an MPB.

An MPB consists of a fixed section and a variable length section. The fixed section contains control information, and the variable length section contains substitution data. The MPB does not contain any message text. Issue TRANMSG to retrieve the message text for this MPB. Issue BLDMPB once for each MPB that you want to construct. Use BLDMPB together with UPDTMPB.

See [z/OS MVS Programming: Assembler Services Guide](http://www.ibm.com/systems/z/os/zos/bkserv/) for more information on using the BLDMPB macro.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state and any PSW key
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: PASN=HASN=SASN or PASN=HASN=SASN
- **AMODE**: 24- or 31-bit
- **ASC mode**: Primary
- **Interrupt Status**: Enabled for I/O and external interrupts
- **Locks**: No locks held
- **Control parameters**: Not applicable

Programming requirements

Before invoking BLDMPB, you must obtain storage for the MPB. You must include the mapping macro CNLMMPB. See [z/OS MVS Data Areas](http://www.ibm.com/systems/z/os/zos/bkserv/) in z/OS Internet Library at [http://www.ibm.com/systems/z/os/zos/bkserv/] for more information on CNLMMPB.

Restrictions

None.

Input register information

Before issuing the BLDMPB 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
BLDMPB macro

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 BLDMPB macro is written as follows:

```
name name

b

BLDMPB

b
```

```
MPBPTR=mpb addr

,MPBLEN=mpb length addr

,MSGID=msg id addr

,MSGIDLEN=msg id length addr

,MSGFMTNM=format num addr

,MSGLNNM=line num addr
```

Parameters

The parameters are explained as follows:

`MPBPTR=mpb addr`
Specifies the address or a register containing the address of the area in which
BLDMPB is to build the MPB.

`,MPBLEN=mpb length addr`
Specifies the address or a register containing the address of the length of the
area in which BLDMPB is to build the MPB. Determine the length by adding the
BLDMPB macro

length of the variable data to the length of the MPB header section. Variable
data includes entries associated with each piece of substitution data.

,MSGID=msg id addr
   Specifies the address or a register containing the address of the area that
   contains the message identifier.

,MSGIDLEN=msg id length addr
   Specifies the address or a register containing the address of the length of the
   MSGID field. The message identifier can be up to 10 characters long. If you
don't specify MSGIDLEN, BLDMPB will use, as a default, the length of the
MSGID field in the DSECT mapping. You must specify MSGIDLEN if you use
register notation for the MSGID keyword.

,MSGFMTNM=format num addr
   Specifies the address or a register pointing to an area containing a 3-byte
   message format number. If you do not specify MSGFMTNM, the default is a
   blank.

,MSGLNNM=line num addr
   Specifies the address or a register pointing to an area containing the 2-byte
   message line number. If you do not specify MSGLNNM, the default is a blank.

Return and reason codes

When BLDMPB completes, register 15 contains a return code, and register 0
contains a reason code:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Successful processing.</td>
</tr>
<tr>
<td>0C</td>
<td>33</td>
<td>The MPB is too small.</td>
</tr>
<tr>
<td>0C</td>
<td>34</td>
<td>The value for MSGIDLEN is zero or negative.</td>
</tr>
</tbody>
</table>

Example

Build and update an MPB for a message that contains as substitution data the third
day of the week.

```
BLDMPBA CSECT
BLDMPBA AMODE 31
BLDMPBA RMODE ANY
STM 14,12,12(13)
BALR 12,0
USING *,12
ST 13,SAVE+4
LA 15,SAVE
ST 15,8(13)
LR 13,15
***********************************************************************
* OBTAIN WORKING STORAGE AREA FOR THE MPB *
***********************************************************************
GETMAIN RU,LV=STORLEN,SP=SP230
LR R4,R1 SAVE GETMAINED AREA ADDRESS
***********************************************************************
* CREATE MPB HEADER SECTION *
***********************************************************************
* BLDMPB MPBPTR=(R4),MPBLEN=MPBL,MSGID=MSGID,
  MSGIDLEN=MIDLEN
***********************************************************************
```
**BLDMPB macro**

```
***********************************************************************
* ADD SUBSTITUTION DATA TO MPB *
***********************************************************************
* LR R2,R4 GET ADDRESS OF GETMAINED STORAGE
A R2,MPBL ADD LENGTH OF MPB TO POINT TO C
    VARIABLE AREA
    USING VARS,R2
* UPDTMPB MPBPTR=(R4),MPBLEN=MPBL,UBOOFST=VARS,
    token=TOKN,TOKENLEN=TOKL,TOKTTYPE=TOKT,
    SUBSDATA=SDATA,SUBSLEN=SDATAL
*
***********************************************************************
* FREE STORAGE AREA *
***********************************************************************
* FREEMAIN RU,LV=STORLEN,SP=SP230,A=(4)
* L 13,SAVE+4
LM 14,12,12(13)
BR 14
DROP
***********************************************************************
```

**Variables**

- `MPBL`: Address of MPB length
- `MSGID`: MSG ID of message represented by MPB
- `MIDL`: Address of MSG ID length
- `TOKN`: Token name
- `TOKL`: Length of token name
- `TOKT`: Token type (DAY)
- `SDATA`: Substitution data (3rd day of week)
- `SDL`: Substitution data length
- `SAVE`: Save area
- `SP230`: Subpool specification for GETMAIN
- `STORLEN`: Length of getmained storage
- `MPBLN`: MPB length
- `R1`: Register 1
- `R2`: Register 2
- `R4`: Register 3

**DSECT**

- `CNLMMPB`
- `VARS DSECT`
- `VARSAREA DS CL24`
- `VARSLEN EQU =VARS`
Chapter 7. BLSABDPL — Map dump formatting exit data

Description

The BLSABDPL macro maps several structures that are part of the interface to dump formatting exits. Dump formatting exits are routines that receive control from one of the following:

- The interactive problem control system (IPCS)
- The SNAP macro or SNAPX macro
- The ABEND macro.

BLSABDPL maps the following structures:

- The processor status record
- The storage access parameter list
- The select ASID parameter list
- The control block and format model processor parameter list
- The ECT parameter list
- The format parameter list extension block.


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

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

```
b
```

One or more blanks must precede BLSABDPL.
Parameters

The parameters are explained as follows:

1. **AMDCPST=YES**,  **AMDCPST=NO**
   Specifies whether the format of the CPU status data, available through the IPCS storage access services, is to be mapped (YES) or suppressed (NO).
   When this parameter is not specified, the default is NO.
   The system uses DSECT AMDCPMAP to map the format of CPU status data, AMDCPST = YES, and ignores the DSECT = NO option when specified.

2. **AMDEXIT=YES**,  **AMDEXIT=NO**
   Specifies whether the common exit parameter list, BLSABDPL, is to be mapped (YES) or suppressed (NO).
   When this parameter is not specified, the default is YES.
   The common exit parameter list contains two parts: ABDPL and ADPLEXTN. DSECT=YES causes DSECT statements to be generated for both. DSECT=NO suppresses the DSECT statements and causes ABDPL and ADPLEXTN to be defined as the labels associated with the first bytes described in the ABDPL and ADPLEXTN exit parameter lists.

3. **AMDOSEL=YES**,  **AMDOSEL=NO**
   Specifies whether the BLSABDPL macro is to be mapped (YES) or suppressed (NO).
   The system uses DSECT AMDCPMAP to map the format of CPU status data, AMDCPST = YES, and ignores the DSECT = NO option when specified.
Specifies whether the select ASID service output data available under IPCS is to be mapped (YES) or suppressed (NO).

When this parameter is not specified, the default is YES.

When the DSECT=NO option is specified, it is ignored. The select ASID parameter list is mapped by DSECT ADPLPSEL.

The system uses DSECT ADPLPSEL to map the select ASID parameter list, AMDOSEL = YES, and ignores the DSECT = NO option when specified.

Specifies whether the storage access service parameter list is to be mapped (YES) or suppressed (NO).

When this parameter is not specified, the default is YES.

The storage access service parameter list is described as ADPLPACC. DSECT=YES causes DSECT statements to be generated for ADPLPACC. DSECT=NO suppresses the DSECT statements and causes ADPLPACC to be defined as the label associated with the first byte described in the storage access service parameter list.

Specifies whether the ECT service parameter list is to be mapped (YES) or suppressed (NO).

When this parameter is not specified, the default is YES.

The ECT service parameter list is described as ADPLPECT. DSECT=YES causes DSECT statements to be generated for ADPLPECT. DSECT=NO suppresses the DSECT statements and causes ADPLPECT to be defined as the label associated with the first byte described in the ECT service parameter list.

Specifies whether the parameter list used by both the control block formatter and the format model processor services is to be mapped (YES) or suppressed (NO).

When this parameter is not specified, the default is YES.

The parameter list used by both the control block formatter and the format model processor services is described as ADPLPFMT. DSECT=YES causes DSECT statements to be generated for ADPLPFMT. DSECT=NO suppresses the DSECT statements and causes ADPLPFMT to be defined as the label associated with the first byte described in the parameter list.

Specifies whether the select ASID service parameter list is to be mapped (YES) or suppressed (NO).

When this parameter is not specified, the default is YES.

The ASID service parameter list is described as ADPLPSEL. DSECT=YES causes the DSECT statements to be generated for ADPLPSEL. DSECT=NO suppresses the DSECT statements and causes ADPLPSEL to be defined as the label associated with the first byte described in the ASID service parameter list.
BLSABDPL macro

\texttt{DSECT=YES}
\texttt{DSECT=NO}

Specifies whether parameter lists mapped by BLSABDPL are to be mapped as DSECTs (YES) or not (NO).

When this parameter is not specified, the default is YES.

\textbf{Note:} Output data from services can also be mapped by BLSABDPL. Output data are always mapped as DSECTs. These DSECTs cannot be suppressed by \texttt{DSECT=NO}. To determine whether \texttt{DSECT=NO} can suppress a specific DSECT, see the above parameters.

\section*{Example}

Code the macros to invoke the select ASID service routine. This routine generates a list of selected address spaces within a dump by reserving space for an initialized select ASID service parameter list and by defining the mapping of the ABDPL for the user-written exit routine.

\begin{verbatim}
BLSABDPL DSECT=NO,AMDEXIT=NO,AMDOSEL=NO,AMDPACC=NO,
          AMDPFM=NO,AMDPFCS=NO,AMDPSEL=YES
\end{verbatim}
Chapter 8. BLSACBSP — Map the control block status (CBSTAT) parameter list

Description

BLSACBSP maps the control block status (CBSTAT) parameter list. Use this parameter list when calling the CBSTAT service from within an installation-written interactive problem control system (IPCS) exit routine.

The control block status (CBSTAT) service invokes all CBSTAT exit routines for a requested control block.


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

```
  name name
  \bsd\bsd
  BLSACBSP
  b

  DSECT=YES
  Default: DSECT=YES
  DSECT=NO
```

name: Symbol. Begin name in column 1.

b: One or more blanks must precede BLSACBSP.

BLSACBSP: One or more blanks must follow BLSACBSP.
BLSACBSP macro

ABITS=31
ABITS=64

Default: ABITS=31

Note: Users must supply a label (name), and start it in column 1 of the BLSACBSP macro. When the BLSACBSP macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

Parameters

The parameters are explained as follows:

DSECT=YES
DSECT=NO

A CBSTAT parameter list is mapped with a DSECT (DSECT=YES) or a CBSTAT parameter list is mapped, but no DSECT is generated (DSECT=NO).

ABITS=31
ABITS=64

Either a 31-bit or a 64-bit storage map is to be referenced.

Example

Code the following to map the CBSTAT service list, but not as a DSECT. All fields will appear with the prefix CBSP:

CBSP  BLSACBSP  DSECT=NO
Chapter 9. BLSADSY — Map the add symptom service parameter list

Description

BLSADSY maps the add symptom service parameter list. Use this parameter list when calling the add symptom service from within an installation-written interactive problem control system (IPCS) exit routine. The add symptom service permits exit routines to generate symptoms from stand-alone dumps, SVC dumps, and the SYSMDUMP type of ABEND dump.

See [z/OS MVS IPCS Customization](http://www.ibm.com/systems/z/os/zos/bkserv/) for information about the add symptom service. See [z/OS MVS Data Areas in z/OS Internet Library at](http://www.ibm.com/systems/z/os/zos/bkserv/) for a mapping of the BLSADSY data area.

Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

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

DSECT=YES

Default: DSECT=YES

DSECT=NO
Note: Users must supply a label (name), and start it in column 1 of the BLSADSY macro. When the BLSADSY macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

Parameters

The parameters are explained as follows:

DSECT=YES
DSECT=NO

An add symptom service parameter list is mapped with a DSECT (DSECT=YES) or an add symptom service parameter list is mapped, but no DSECT is generated (DSECT=NO).

Example

Code the following to map the add symptom service list, but not as a DSECT. All fields will appear with the prefix ADSY.

ADSY  BLSADSY DSECT=NO
Chapter 10. BLSAPCQE — Map the contention queue element (CQE) create service parameter list

Description

BLSAPCQE maps the contention queue element (CQE) create service parameter list. Use this parameter list when calling the CQE create service from within an installation-written interactive problem control system (IPCS) exit routine to create CQE entries in the dump directory.


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

```plaintext
name name

/bslash

BLSAPCQE

/bslash

DSECT=YES

DSECT=YES

Default: DSECT=YES
```

© Copyright IBM Corp. 1988, 2012
BLSAPCQE macro

**Note:** Users must supply a label (*name*), and start it in column 1 of the BLSAPCQE macro. When the BLSAPCQE macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

**Parameters**

The parameters are explained as follows:

- **DSECT=YES**
  - A contention queue element (CQE) create parameter list (PCQE) is mapped with a DSECT (DSECT=YES) or a PCQE is mapped, but no DSECT is generated (DSECT=NO). IPCS initializes the PCQE as follows:
    - The PCQE control block identifier is filled in.
    - Pointer and length fields are set to 0.
    - The OWNER/WAITER identifier field is set to “O ” to indicate an owner CQE.
    - Character fields not specifically mentioned are set to blanks.
    - The data description of the control block which represents the owner or waiter for the resource is set as follows:
      - Address space type code is set to indicate a virtual address (CV). This is the only address space type code allowed.
      - The processor field is set to X'FFFFFFFF'. This is done to avoid specifying processor 0 accidentally.
      - The ASID is set to 1 for the MASTER address space. The ASID field needs to get set to the ASID for the owner or waiter for the resource. If the ASID is not known and the control block is in common storage, use ASID 1 as the default.
      - The data type is set to “M” to indicate that the specified name is a STRUCTURE. This is the only data type allowed for this release.

- **DSECT=NO**

**Example**

Code the following to map the contention queue element (CQE) create service parameter list, but not as a DSECT. All fields will appear with the prefix PCQE.

```
PCQE BLSAPCQE DSECT=NO
```
Chapter 11. BLSQFXL — Map the format exit routine list (FXL)

Description
BLSQFXL maps the format exit routine list (FXL) used by model processor formatting exit routines. FXL contains the addresses of data of potential interest to the model processor formatting exit routine, as well as a description of the formatted line.


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

Programming requirements
None.

Restrictions
None.

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

Performance implications
None.

Syntax
The standard form of the BLSQFXL macro is written as follows:

```
name name

/bslash

BLSQFXL must begin in column 1.

b

One or more blanks must follow BLSQFXL.

DSECT=YES

Default: DSECT=YES

DSECT=NO
```

© Copyright IBM Corp. 1988, 2012
Parameters

The parameters are explained as follows:

- **DSECT=YES**
- **DSECT=NO**

  An FXL is mapped with a DSECT (DSECT=YES) or an FXL is mapped, but no DSECT is generated (DSECT=NO).

Example

Code the following to map an FXL, but not as a DSECT.

```
BLSQFXL  DSECT=NO
```
Chapter 12. BLSQMDEF — Define a control block format model

Description

The BLSQMDEF macro starts and ends the definition of a control block format model. The end of the model is indicated by a BLSQMDEF macro with only the END keyword specified.

The BLSQMDEF and BLSQMFLD macros work together to create a formatting model. This is the structure of the formatting model:

- One BLSQMDEF macro to begin the model definition.
- At least one BLSQMFLD macro to define the attributes of a desired control block field.
- One BLSQMDEF macro to end the model definition.

The order of the BLSQMFLD statements in the formatting model determines the order of the fields in the output of the formatting process. Only the BLSQMFLD macro can be placed between the BLSQMDEF macros that delimit the start and end of the model definition. Use the BLSQSHDR macro, which defines text strings to be displayed in the formatted output, to clarify the data. Place BLSQSHDR after the second BLSQMDEF.

BLSQMDEF, BLSQMFLD, and BLSQSHDR allow interactive problem control system (IPCS) and SNAP users to specify the presentation of data and messages produced by user-written exit routines.

See [z/OS MVS IPCS Customization](https://www.ibm.com/support/knowledgecenter/SSEPGG_2.2.0/com.ibm.zos.mvs/zos_zosipc_customization.html) for information about format models.

Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

name

name: Symbol. Begin name in column 1

b

One or more blanks must precede BLSQMDEF.

BLSQMDEF

b

One or more blanks must follow BLSQMDEF.

END

END is required when the BLSQMDEF macro is terminating the current format model definition. When END is specified, no other options are allowed.

,BASELBL=label

label: Symbol.

,CBLen=value

value: Decimal constant, hexadecimal constant, or an absolute value. CBLen is required unless the END parameter is specified.

,MAINTLV=name

name: 1 to 8 byte character string.

,ACRONYM=name

name: 1 to 8 byte character string
When ACRONYM is specified, the ACROLBL or ACROFF parameters must also be specified. When neither ACROLBL nor ACROFF are specified, a default zero is assumed.

,ACROLEN=value

value: Decimal constant, hexadecimal constant, or absolute expression of a number from 1 to 8.

,ACROLBL=label

label: Symbol.
Use ACROLBL only when BASELBL is specified.

,ACROFF=value

value: Decimal constant, hexadecimal constant, or absolute value.
Use ACROFF when ACRONYM is not at offset zero and BASELBL is not specified, or when both ACROFF and ACROLBL are specified.

,PREFIX=value

value: Integer constant 0 - 8 inclusive.
Default: PREFIX=3

,OFFSETS=PRINT

Default: OFFSETS=PRINT

,OFFSETS=NOPRINT

,STRTCOL=value

value: Decimal constant, hexadecimal constant, or an absolute expression.
Default: STRTCOL=0

,LBLSPC=value

value: Decimal constant, hexadecimal constant, or an absolute expression.
Default: LBLSPC=0

,HEADER=name

name: One to eight byte character string.
When HEADER is not specified, ACRONYM value is used.
When neither HEADER nor ACRONYM is specified, only the virtual address of the block is displayed as a header.

,VIEWMATCH=VALUE

Default: View matching by value.
Parameters

The parameters are explained as follows:

END
Specifies the termination of the control block model. This parameter is required to end the control block definition. When this parameter is specified all other parameters are ignored.

,BASELBL=label
Specifies the label of an assembler statement, used to calculate field offsets. When specified, all field offsets calculated by the BLSQMFLD macro are relative to this label. When not specified, all field offsets must be explicitly specified on the BLSQMFLD macro via the OFF parameter.

,CBLLEN=value
Specifies the total length of the control block. Value may be a decimal constant, a hexadecimal constant, or an absolute expression of a number from 0 to 32767. If a length of zero is specified, the length of the control block must be separately specified when the model is used. This parameter is required except when the END parameter is specified. This value is used when the format model processor service accesses the data from the dump on behalf of the calling exit program.

,MAINTLV=name
Specifies the maintenance level of the control block. The maintenance level name may be a 1 to 8 byte character string that contains no blanks.

,ACRONYM=name
Specifies the contents of the control block acronym field. Name may be a one to eight byte character string that contains no blanks. When this field is specified, the ACROBL or ACROFF parameter should also be specified to define the offset of the acronym field within the control block. When neither the ACROBL nor the ACROFF parameter is specified, an offset of zero is assumed. The model processor service compares the contents of the data at the specified offset and length with this name when the calling exit program requests the option to check acronyms. The name is also used to form the dump header when the header keyword is not coded.

,ACROLEN=value
Specifies the length of the acronym name, defined by the ACRONYM parameter, when the acronym name requires blanks. When omitted, the length is the actual length of the name specified in the ACRONYM parameter without blanks. Value may be a decimal constant, hexadecimal constant, or absolute expression of a number from zero to eight.

,ACROBL=label
Specifies the label on the assembler statement that defines the acronym field. This label is used with the label provided by BASELBL to calculate the acronym field offset. Use this parameter only when BASELBL is specified. The ACROBL parameter is ignored when ACROFF is specified.

,ACROFF=value
Specifies the offset of the field containing the control block acronym within the control block. Use this parameter when the acronym is not at offset zero and BASELBL is not specified. Value may be a decimal constant, hexadecimal constant, or absolute expression.
BLSQMDEF macro

\textbf{\texttt{,PREFIX=value}}

Specifies the number of characters to be removed from the front of a field name to produce the field label. The field name is defined by the NAME parameter of the BLSQMFLD macro. Value must be an integer constant 0 - 8. When PREFIX=8 is specified, the fields have no labels, and the model processor service does not allocate print buffer space for labels. This is called "no-label mode" and is used to produce denser data output. When not specified, the default is PREFIX=3. PREFIX may be re-specified on a succeeding BLSQMFLD macro.

\textbf{\texttt{,OFFSETS=PRINT}}
\textbf{\texttt{,OFFSETS=NOPRINT}}

Specifies whether the field offset information should be printed at the beginning of each output line of the formatted control block. PRINT specifies that offset information should be included on the formatted line; NOPRINT causes the offset information to be suppressed. When this parameter is not specified, a default of PRINT is used. Specifying OFFSETS=NOPRINT is identical to setting bit ADPLPSOF in field ADPLPOPT to B'1'; both the OFFSETS=NOPRINT parameter and the ADPLPSOF bit suppress offsets. ADPLPOPT is mapped by the BLSABDPL macro.

\textbf{\texttt{,STRTCOL=value}}

Specifies a left margin for each line of the formatted control block. Value may be a decimal constant, a hexadecimal constant, or an absolute expression. When not specified, or specified as zero, the format model processor uses the value specified by IPCS or SNAP in the field ADPLSCOL in ADPLEXTN, which is mapped by the BLSABDPL macro.

\textbf{\texttt{,LBLSPC=value}}

Specifies the spacing between label fields in the formatted output. Value may be a decimal constant, hexadecimal constant, or an absolute expression. When not specified, or specified as zero, it indicates to the format model processor that the value specified by IPCS or SNAP should be used in field ADPLCOLS in ADPLEXTN, which is mapped by the BLSABDPL macro. The LBLSPC value is initially set to 20.

\textbf{Note:} When value is 18, the output is condensed.

\textbf{\texttt{,HEADER=name}}

Specifies the heading that precedes the formatted control block. The heading consists of either the HEADER or ACRONYM followed by the virtual address of the block. Name may be any one to eight byte character string that contains no blanks. When HEADER is omitted, the ACRONYM value is used for the heading. When neither the ACRONYM parameter nor the HEADER parameter is specified, the formatted control block has the virtual address as a heading.

\textbf{\texttt{,VIEWMATCH=VALUE}}

Specifies that the first eight bits of the view control fields in the format parameter and a model entry must match to process that model entry. When not specified, any bit match in the first 12 bits is sufficient. Rules for the component portion of the view apply in both cases.
Chapter 13. BLSQMFLD — Specify a formatting model field

Description

The BLSQMFLD macro identifies fields that are to be formatted. These fields are within a data area or a control block. A BLSQMFLD macro must be coded for each field.

The BLSQMDEF and BLSQMFLD macros work together to create a formatting model for a control block. This is the structure of the model:

- One BLSQMDEF macro to begin the model definition.
- At least one BLSQMFLD macro to define the attributes of a desired control block field.
- One BLSQMDEF macro to end the model definition.

The order of the BLSQMFLD statements in the formatting model determines the order of the fields in the output of the formatting process. Only the BLSQMFLD macro can be placed between the BLSQMDEF statements. The BLSQSHDR macro, which defines text strings to be displayed in the formatted output, clarifies the data and should be placed after the second BLSQMDEF.

BLSQMDEF, BLSQMFLD, and BLSQSHDR allow interactive problem control system (IPCS) and SNAP users to specify the presentation of data and messages produced by user-written exit routines.

See \[z/OS MVS IPCS Customization\] for information about format models.

Environment

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

Programming requirements

None.

Restrictions

Register information

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

Performance implications

None.

Syntax

This is the standard form of the BLSQMFLD macro:

\[\text{name name: Symbol. Begin name in column 1.}\]

\[b\]

One or more blanks must precede BLSQMFLD.
BLSQMFLD macro

One or more blanks must follow BLSQMFLD.

NAME=label
NAME=*  

,SHDR=addr  
  addr: A-type address.
  Note: When SHDR is specified, only CALLRTN, NEWLINE, NOSPLIT, and VIEW are allowed.

,OFF=value  
  value: Decimal constant, hexadecimal constant, or absolute value.
  Note: OFF is required when BASELBL is not specified on the BLSQMDEF macro or when NAME=* is specified on the BLSQMFLD macro.

,LEN=value  
  value: Decimal constant, hexadecimal constant, or absolute expression.
  Note: LEN is required when name parameter label is unresolved.

,VIEW=(list)  
,VIEW=value  
  (list): Integers between 1 and 16, inclusive.
  value: Decimal constant, hexadecimal constant, or absolute value.
  Default: VIEW=X'0200'.

,ARRAY=constants  
,ARRAY=DL1,DU1,DL2,DU2  
constants: ((DL1,DU1),(DL2,DU2))

,ARRAY=value  
  DL1,DL2,DU1,DU2: Decimal constants, hexadecimal constants, or absolute values.

,ARRAY=*  
,ARRAY=END  
  value: Decimal constant, hexadecimal constant, or absolute value.
  Note: LEN and OFF are ignored when the specification of ARRAY= is other than ARRAY=END.
  END terminates an array definition.

,DTYPE=ANY
,DTYPE=QANY
,DTYPE=HEX
,DTYPE=EBCDIC

,DECODE
,INVERT
,ATTACH
,IMBED
,STACK
,CALLCBF
,NEWLINE
,NOLABEL
,CALLRTN
BLSQMFLD macro

\texttt{,PREFIX=\textit{value}} \quad \textit{value}: Integers between 0 and 8.

\textbf{Note}: When omitted, \textit{value} specified in the last preceding BLSQMDEF or BLSQMFLD macro is used.

\texttt{,NOSPLIT}

\texttt{,NUMDEC} \quad \textbf{Default}: Hexadecimal.

\texttt{,NOCOLNM} \quad \textbf{Default}: Number the columns.

\texttt{,NOROWNM}

\texttt{,STRTCOL=\textit{value}} \quad \textit{value}: Decimal constant, hexadecimal constant, or absolute value.

\textbf{Default}: Value specified by IPCS or SNAP.

\texttt{,COLNUM=\textit{value}} \quad \textit{value}: Decimal constant, hexadecimal constant, or absolute value.

\textbf{Default}: A value is calculated.

\texttt{,COLSEP=\textit{value}} \quad \textit{value}: Decimal constant, hexadecimal constant, or absolute value.

\textbf{Default}: A value is calculated.

\texttt{,ITEMSEP=\textit{value}} \quad \textit{value}: Decimal constant, hexadecimal constant, or absolute value.

\textbf{Default}: A value is calculated.

\texttt{,ORDER=(1,2)} \quad \textbf{Default}: ORDER=(1,2)

\texttt{,ORDER=(2,1)}

\texttt{,HEXONLY}

\texttt{,MODELNAME(modelname)}

\texttt{,SHDR=\textit{addr}}

\texttt{,MSGID(msgid)}

\texttt{,SRCNDX} \quad \textbf{Default}: 0

---

**Parameters**

The parameters are explained as follows:

\texttt{NAME=label}

\texttt{NAME=*}

\textit{NAME *} \quad \textbf{Specifies} the name of the control block field described by the BLSQMFLD macro. When BASELBL is specified on the BLSQMDEF macro, the NAME label is used with the BASELBL label to calculate the offset of this field from the start of the control block. When BASELBL is not specified on the BLSQMDEF macro, OFF is required on the BLSQMFLD macro.

A single asterisk specifies an unnamed, reserved field. The use of single asterisk for the name of a control block field requires that the OFF and LEN parameters be specified. The format model processor service replaces the asterisk with a “RSV.....” label.

\texttt{,SHDR=\textit{addr}}

\textit{SHDR * } \quad \textbf{Specifies} the address of a character string used as a subheading in the control
BLSQMFLD macro

block format. The address must be valid in an assembler A-type DC instruction. This parameter should point to a one-byte length field followed by the heading character string. The length byte indicates the length of the heading string and not the length of the length byte. The BLSQSHDR macro is used to define subheaders.

When this parameter is specified, only CALLRTN, NEWLINE, NOSPLIT, and VIEW can be specified. Other parameters are ignored.

,OFF=value

Specifies the offset of the field from the beginning of the control block. The value can be a decimal constant, a hexadecimal constant, or an absolute expression. When this parameter is specified, the value defined overrides the default field offset generated by the NAME label on this macro and the BASELBL label on the BLSQMDEF macro.

OFF is ignored when the specification of ARRAY= is other than ARRAY=END.

This parameter is required when the BASELBL parameter is not specified on the BLSQMDEF macro or when NAME=* is specified on the BLSQMFLD macro.

,LEN=value

Specifies the length of the control block field. The value is a decimal constant, hexadecimal constant, or absolute expression of a number from 1 to 32767. This parameter is required when no data constants with a label exist in the assembly program as defined by the NAME parameter, or when use of the assembler length attribute would not result in a correct length determination for the data constant representing the field.

LEN is ignored when specification of ARRAY= is other than ARRAY=END.

An assembly error occurs when LEN is not specified and there is no assembler statement with a label matching the one specified by NAME.

,VIEW=(list)

,VIEW=value

Specifies up to sixteen different views of the control block fields. Any combination of one to sixteen view attributes can be specified for each field. The caller of the model processor exit service provides a view pattern defining the views to be formatted. The view field consists of a twelve-bit general view followed by a four-bit component view. When the component view in a model entry is zero, any matching bit causes that model entry to be processed. When the component view in the model is not zero, there must be a matching bit in both the general and component view fields.

When VIEWMATCH=VALUE is coded on the first BLSQMDEF macro of the model, the model processor compares the first byte of the two view fields and requires an exact match to process the model entry. This feature is convenient for decoding a value-coded byte.

The list is an unordered list of attributes; each attribute can be a decimal integer between 1 and 16, (VIEW=1,2,...,16), binary constant (VIEW=B'0010000000000000'), or hexadecimal constant (VIEW=X'0080').

This chart illustrates the view parameter's control block field options provided through the specification of a 4-digit hexadecimal number. Any combination of the view fields listed can be specified.
## BLSQMFLD macro

<table>
<thead>
<tr>
<th>Hexadecimal Code</th>
<th>User-defined fields to be displayed</th>
</tr>
</thead>
<tbody>
<tr>
<td>x'8000'</td>
<td>keyfield</td>
</tr>
<tr>
<td>x'4000'</td>
<td>summary field</td>
</tr>
<tr>
<td>x'2000'</td>
<td>register save area</td>
</tr>
<tr>
<td>x'1000'</td>
<td>linkage field</td>
</tr>
<tr>
<td>x'0800'</td>
<td>error fields</td>
</tr>
<tr>
<td>x'0400'</td>
<td>hexadecimal dump</td>
</tr>
<tr>
<td>x'0200'</td>
<td>non-reserved field</td>
</tr>
<tr>
<td>x'0100'</td>
<td>reserved fields</td>
</tr>
<tr>
<td>x'0080'</td>
<td>static array or decode flag fields</td>
</tr>
<tr>
<td>x'0040'</td>
<td>dynamic array</td>
</tr>
<tr>
<td>x'0020'</td>
<td>input field</td>
</tr>
<tr>
<td>x'0010'</td>
<td>output field</td>
</tr>
</tbody>
</table>

When this parameter is not specified, the default value of VIEW=X'0200'. is used. See [z/OS MVS IPCS User's Guide](https://www.ibm.com/) and [z/OS MVS IPCS Commands](https://www.ibm.com/) for more information about ADPLPFMT.

```plaintext
,ARRAY=((DL1,DU1),(DL2,DU2))
,ARRAY=value
,ARRAY=*  
,ARRAY=END
```

Specifies that the succeeding BLSQMFLD statements define a set of fields that are repeated in the control block.

The ARRAY parameter on the BLSQMFLD macro indicates that the BLSQMFLD macro is the beginning or the end of an array definition.

The LEN and OFF parameters are ignored when specification of ARRAY= is other than ARRAY=END.

The VIEW specified applies to all fields within the array. The VIEW specified on the BLSQMFLD macro that starts an array should be the composite of the VIEW on all fields within the array.

When ARRAY=((DL1,DU1),(DL2,DU2)) is coded, a two dimensional array is specified. DL1 is the lower limit of the first dimension and DU1 is the upper limit of the first dimension. DL2 is the lower limit of the second dimension and DU2 is the upper limit of the second dimension. When a lower limit for a dimension is not specified, the default is 1. No default exists for the upper limit of a dimension. An asterisk (*) can be coded for either the upper limit or lower limit of the dimension to indicate that the dimension is to be provided by the calling program at execution time in fields ADPLPDL1, ADPLPDL2, ADPLPDU1, ADPLPDU2 in the format parameter.

The total length of an array element must be accounted for in the total of the LEN values of the fields within the array definition. VIEW=0 can be coded on fields within the array that are never to be displayed.

**Notes:**

1. The correspondence of a dimension to a row or column is determined by the ORDER parameter.
2. When the array is larger than 65,535 bytes, the calling program must process the array in sections. The formatter equates the lower limit for each dimension to the value one to address the array entries in a buffer. It uses the specified values to number rows and columns in the formatted output.

The format parameter extension is used to define blocks of storage of arbitrary length to eliminate this restriction.

When ARRAY=\textit{value} is coded, a one dimensional array (list) is specified. \textit{Value} defines the number of array entries contained in the control block.

When ARRAY=\textit{*} is coded, the number of entries in the one-dimensional array (list) are to be provided by the calling program at execution time in the ADPLPDAC field of the format parameter.

The total length of an array element must be accounted for in the total of the LEN values of the fields within the array definition. VIEW=0 can be coded on fields within the array that are never to be displayed.

When ARRAY=END is coded, the array definition is terminated.

\begin{itemize}
  \item \texttt{DTYPE=HEX}
  \item \texttt{DTYPE=EBCDIC}
  \item \texttt{DTYPE=ANY}
  \item \texttt{DTYPE=QANY}
\end{itemize}

\texttt{DTYPE=HEX} indicates that the area to be displayed contains four-bit hexadecimal digits. \texttt{DTYPE=EBCDIC} indicates that the area to be dumped contains eight-bit EBCDIC characters.

\texttt{DTYPE=ANY} specifies that the data is either EBCDIC or hexadecimal. When the data is EBCDIC, the model processor treats the data as EBCDIC. When any of the data is not EBCDIC, the model processor treats all the data as four-bit hexadecimal digits. The field must be less than 256 bytes.

\texttt{DTYPE=QANY} specifies that the next entry in the model is a subheader entry, with a view field of all zero. The text of the subheader is interpreted to be the value or values that are treated as character data. When the subheader is shorter than or equal to the length of the data field, a comparison is made using the subheader length. When the subheader is longer than the data field, the subheader length must be a multiple of the data field length, and multiple comparisons are made.

In both cases, the EBCDIC version is presented in four-byte segments unless NOSPLIT is also coded.

\begin{itemize}
  \item \texttt{DECODE}
\end{itemize}

\texttt{DECODE} specifies that the model entry describes one of these decoding operations:
\begin{itemize}
  \item Flag field decoding
  \item Format imbedded block
  \item Format attached block
  \item Format stacked block
\end{itemize}

When MODELNAME is coded, the NAME field is copied into the model entry. Otherwise, it is treated as the address of the named model. The CALLCBF parameter also specifies whether the name is to be interpreted as a model name or as an acronym. When acronym, the system calls the control block formatter.

\texttt{DECODE} is not supported within an array.

\begin{itemize}
  \item \texttt{INVERT}
\end{itemize}

\texttt{INVERT} specifies that the flag field is to be inverted and that decoding is to be
performed according to a model. The flag fields are inverted to allow the decoding of flags that are in the zero state. To specify flag fields use the offset and length parameters. The flag field can be up to four bytes long. The model is described by the label field and the control flags. INVERT is valid only when DECODE is specified, and decoding is performed only when the views match.

**,ATTACH**

Specifies that the dump data referenced by a pointer at the offset, specified by the offset parameter, is to be formatted according to a format model. When the length parameter value is other than zero or four, that value is added to the value of the pointer. The model is described by the name field and the control flags. ATTACH is valid only when DECODE is specified, and formatting occurs only when the views match.

**,IMBED**

Specifies that the data identified by the offset and length parameters is to be formatted according to a model. The model is described by the name field and control flags. The starting offset is the offset in the containing block, and the header is suppressed. Data areas formatted appear to be part of the containing block. IMBED is valid only when DECODE is specified, and formatting occurs only when the views match. Register save areas in control blocks are examples of embedded blocks.

**,STACK**

Specifies that the data identified by the offset and length parameters is to be formatted according to a model. The model is described by the name field and control flags. The starting offset is zero, and the header is not suppressed. Areas formatted are recognizable as distinct entities. STACK is valid only when DECODE is specified. The SDWA in dump header records illustrates the function of STACK.

**,CALLCBF**

Specifies that the name field is an acronym. CALLCBF and MODELNAME are valid only when DECODE is specified in conjunction with ATTACH, IMBED, or STACK. Formatting is performed according to a model, and each ACRONYM corresponds to a particular model. When CALLCBF is not specified, the model processor is called directly.

**,NEWLINE**

Specifies that the field should be printed on the next line of output.

**,NOLABEL**

Specifies that the field label is not to be printed. NAME is required.

**,CALLRTN**

Specifies that the model processor calls the model processor formatting exit after the output line, containing this field, is formatted before it is printed. The model processor formatting exit entry point address is specified by the caller in the parameter list, ADPLPLME, when the model processor is invoked.

**,PREFIX=value**

Specifies the number of characters to be removed from the front of a field name to produce the field label. The field name is defined by the NAME parameter. Value must be an integer constant greater than or equal to zero and less than or equal to eight. When PREFIX is omitted from the current BLSQMFLD macro, the value specified on the last preceding BLSQMFLD or BLSQMDEF macro is used. The BLSQMDEF macro, used to start a model definition, can also be used to set the value of PREFIX. When PREFIX=8 is coded with BLSQMDEF, the model processor operates in “no-label mode”, and does not allocate print buffer columns to labels.
BLSQMFLD macro

,NOSPLIT
   Specifies that the model processor attempts to print all the field data on the
   same output line. When the data does not fit on the current output line, but fits
   on a single output line, the model processor skips to a new line prior to printing
   the data field.
   When NOSPLIT is coded with ANY or QANY, the character string is displayed
   as is, not in four-byte segments. The display might differ when the field is
   treated as hexadecimal.

[NUMDEC
   Specifies that the columns and rows of a two-dimensional array be numbered in
decimal. The default is hexadecimal.

,NOCOLNM
   Specifies that column numbers (headers) of a two-dimensional array be
   suppressed. The default is to number the columns. The NUMDEC parameter
   controls the numbering system used for numbering the columns.

,NOROWNM
   Specifies that the row numbers of a two-dimensional array are to be
   suppressed. The default is to specify the row numbers. NUMDEC parameter
   controls the numbering system used to number the rows. NOROWNM is valid
   only with ARRAY=((value,value),(value,value)).

,STRTCOL=value
   Specifies the left margin of the formatted output. Value indicates the number of
   blanks before the first character. STRTCOL applies only to two-dimensional
   arrays. This specification overrides the value defined by the STRTCOL
   parameter in the BLSQMDEF macro, or by IPCS or SNAP, for the duration of
   displaying the array. When not specified, a default of zero is provided and the
   formatter uses the value specified by the host.

,COLNUM=value
   Specifies the number of columns of a two dimensional array that are to be
   displayed in each line of output. When not specified, or when the specified
   number of columns does not fit in the currently available print buffer, the
   formatter calculates a value consistent with, and not exceeding, the maximum
   line length specified by IPCS or SNAP.

,COLSEP=value
   Specifies the number of blanks to be placed between the columns of a
   two-dimensional array. The default is zero. The model processor uses a
   calculated value.

,ITEMSEP=value
   Specifies the number of blanks to be placed between items within an array
   entry. An array entry can be a structure, and each element of the structure is
   referred to as an “item”. When the array entry is a single item, value is ignored.
   When ITEMSEP is not specified, a default of zero is provided, and the model
   processor uses a calculated value.

,ORDER=(1,2)
   ,ORDER=(2,1)
   Specifies the order in which the data of a two-dimensional array is to be
   processed. When ORDER=(1,2) is specified, the data is processed in
   consecutive rows. When ORDER=(2,1) is specified, the data is processed in
   consecutive columns. The default is ORDER=(1,2).

,HEXONLY
   Specifies that the data is to be displayed in hex. When HEXONLY is omitted,
the data is displayed in both hex and EBCDIC, on the same line, with vertical bars bounding the EBCDIC portion of the display. HEXONLY is valid only when the view parameter specifies X'0400'. This requests a hexadecimal dump.

### MODELNAME(modelname)

Specifies the name of the model to be used in a decoding operation, or the acronym of the data area when CALLCBF is coded.

### MSGID(msgid)

Specifies that the subheader is a message with a message ID that may be conditionally stripped. MSGID is only valid when SHDR is specified.

### SRCNDX

Specifies which one of the 16 entries in the array of buffer and ES addresses is to be used in processing the field. The array is zero origin, so 0 refers to the first and 15 refers to the last. The srcndx value stays in effect for subsequent BLSQMFLD macro invocations until it is changed. This keyword is part of the multiple source formatting feature. The default is zero.

### Example 1

Code the macros that establish a control block formatting model to be used by the model processor to format functional recovery routines (FRRs).

**IEAVTRP3**

```
IEAVTRP3 CSECT

    BLSQMDEF CBLEN=X'0320',MAINTLV=HBB2102,PREFIX=4,OFFSETS=PRINT,X
    HEADER=FRRS
    BLSQMFLD NAME=FRRSEMP,OFF=X'0000',LEN=4,VIEW=X'0202'
    BLSQMFLD NAME=FRRSLAST,OFF=X'0004',LEN=4,VIEW=X'0202'
    BLSQMFLD NAME=FRRSELEN,OFF=X'0008',LEN=4,VIEW=X'0202'
    BLSQMFLD NAME=FRRSCURR,OFF=X'000C',LEN=4,VIEW=X'0202'
    BLSQMFLD NAME=FRRSRSA,OFF=X'0010',LEN=24,VIEW=X'0200'
    BLSQMFLD SHDR=RTM1WA,VIEW=X'0200',NEWLINE
    BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
    BLSQMFLD SHDR=ENTEXT,VIEW=X'0200',NEWLINE
    BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
    BLSQMFLD NAME=FRRSXSTK,VIEW=X'0200',ARRAY=16,NOLABEL
    BLSQMFLD NAME=FRRSXM,OFF=X'00A0',LEN=2,VIEW=X'0200',NEWLINE
    BLSQMFLD NAME=FRRSSAX,OFF=X'00A2',LEN=2,VIEW=X'0200'
    BLSQMFLD NAME=FRRSPAS,OFF=X'00A4',LEN=2,VIEW=X'0200'
    BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
    BLSQMFLD SHDR=ENTS,VIEW=X'0200',NEWLINE
    BLSQMFLD SHDR=BLANK,VIEW=X'0200',NEWLINE
    BLSQMFLD NAME=FRRSENTS,VIEW=X'0200',ARRAY=16,NOLABEL
    BLSQMFLD NAME=FRRSFRRA,OFF=X'0120',LEN=4,VIEW=X'0200',NEWLINE
    BLSQMFLD NAME=FRRSFLGS,OFF=X'0124',LEN=4,VIEW=X'0200'
    BLSQMFLD NAME=FRRSPARM,OFF=X'0128',LEN=24,VIEW=X'0200', ARRAY=END
    BLSQMFLD END
```

**BLANK**

```
BLANK BLSQSHDR ' '
```

**ENTEXT**

```
ENTEXT BLSQSHDR 'FRR ENTRY EXTENSIONS'
```

**ENTS**

```
ENTS BLSQSHDR 'FRR ENTRIES'
```

**RTMIWA**

```
RTMIWA BLSQSHDR 'RTM1 WORK AREA FOLLOWS FRR ENTRIES'
```

### Example 2

Code the macros that establish a control block formatting model to be used by the model processor to format a STAE control block (SCB).

**IEAVTRP4**

```
IEAVTRP4 CSECT

    BLSQMDEF CBLEN=X'0018',MAINTLV=JBB2125,PREFIX=3,OFFSETS=PRINT,X
    HEADER=SCB
    BLSQMFLD NAME=SCBCHAIN,OFF=X'0000',LEN=4,VIEW=X'0200'
    BLSQMFLD NAME=SCBEXIT,OFF=X'0004',LEN=4,VIEW=X'0200'
    BLSQMFLD NAME=SCBEIXT,OFF=X'0010',LEN=4,VIEW=X'0200'
    BLSQMFLD NAME=SCBFLGS,OFF=X'0014',LEN=4,VIEW=X'0200'
    BLSQMFLD NAME=SCBSPARM,OFF=X'0018',LEN=24,VIEW=X'0200', X
```

Chapter 13. BLSQMFLD — Specify a formatting model field  95
Example 3

Define the format of a simple control block. Note that this can be done by using a macro-invocation.

```assembly
MYBLK   DSECT , My simplest control block ever
MYBLKABC DC C'ABC' Identifier
MYBLKDEF DC X'00' Flags
MYBLK00 EQU X'80' 1st flag bit
MYBLK40 EQU X'04' 2nd flag bit
MYBLKGHI DC V(MYENTRY) Address of my program
MYBLKEND EQU * End of my control block
```

Define enough storage to get the block displayed. Note that no ENTRY statement is required for access to CBMODEL1 from other CSECTs since CBMODEL1 lies at the origin of the CSECT.

```assembly
TITLE 'CBMODEL1--Basic Control Block Model'
CBMODEL CSECT , Start definition of simple model
CBMODEL1 BLSQMDEF BASELBL=MYBLK,CBLEN=MYBLKEND-MYBLK,PREFIX=5
   BLSQMFLD NAME=MYBLKABC
   BLSQMFLD NAME=MYBLKDEF
   BLSQMFLD NAME=MYBLKGHI
   BLSQMDEF END End definition of simple model
```

Add acronym checking, the display of the acronym in EBCDIC, and descriptive header for the display in the dump.

```assembly
TITLE 'CBMODEL2--More Elaborate than 1st Model'
ENTRY CBMODEL2 Permit access from other CSECTs
CBMODEL2 BLSQMDEF BASELBL=MYBLK,CBLEN=MYBLKEND-MYBLK,PREFIX=5, X
   ACRONYM=ABC,ACROLBL=MYBLKABC, Acronym field data
   HEADER=MYBLOCK Heading for block in dump
   BLSQMFLD NAME=MYBLKABC,DTYPE=EBCDIC Show it as EBCDIC data
   BLSQMFLD NAME=MYBLKDEF
   BLSQMFLD NAME=MYBLKGHI
   BLSQMDEF END End definition of alternate model
```

Example 4

Assume that the data is stored in this sequence:

```
00010001
00010002
00010003
00010004
00020001
00020002
00020003
00020004
00030001
00030002
00030003
```
And you want the data to be formatted like this:

```
---01--- ---02--- ---03--- ---04---
ARRENTRY ARRENTRY ARRENTRY ARRENTRY
```

```
001 00010001 00010002 00010003 00010004
002 00020001 00020002 00020003 00020004
003 00030001 00030002 00030003 00030004
004 00040001 00040002 00040003 00040004
005 00050001 00050002 00050003 00050004
006 00060001 00060002 00060003 00060004
007 00070001 00070002 00070003 00070004
008 00080001 00080002 00080003 00080004
009 00090001 00090002 00090003 00090004
010 00100001 00100002 00100003 00100004
```

Code the macro that creates a formatting model to do the following:
- Number rows 1 through 10.
- Number columns 1 through 4.
- Use the decimal numbering system for numbering rows and columns.
- Place data in to the array row by row.
- Put one blank between each column.
- Display 4 columns in each group.
- Start printing in the second column from the left margin.
- View all non-reserved fields.
- Print the field label ARRENTRY.

One way to code the macro:

```
BLSQMFLD NAME=ARRAYX,ARRAY=((1,10),(1,4)),VIEW=X'0200', X
   STRTCOL=1,COLSEP=1,COLNUM=4,NUMDEC,NOLABEL
BLSQMFLD NAME=ARRENTRY,OFF=0,LEN=4,ARRAY=END,VIEW=X'0200'
```

**Example 5**

Assume that the data is stored in this sequence:

```
00010001
00010002
00010003
00010004
00020001
00020002
00020003
00020004
00030001
00030002
00030003
00030004
   .
   .
   .
00090001
```
And you want the data to be formatted this way:

```
---05--- ---06--- ---07--- ---08--- ---09---
  ARRENTRY ARRENTRY ARRENTRY ARRENTRY ARRENTRY
-------- -------- -------- -------- --------
 000  00100001  00020001  00030001  00040001  00050001
 001  00100002  00020002  00030002  00040002  00050002
 002  00100003  00020003  00030003  00040003  00050003
 003  00100004  00020004  00030004  00040004  00050004
---0A--- ---0B--- ---0C--- ---0D--- ---0E---
  ARRENTRY ARRENTRY ARRENTRY ARRENTRY ARRENTRY
-------- -------- -------- -------- --------
 000  00100001  00020001  00030001  00040001  00050001
 001  00100002  00020002  00030002  00040002  00050002
 002  00100003  00020003  00030003  00040003  00050003
 003  00100004  00020004  00030004  00040004  00050004
```

Code the macro that creates a formatting model to do the following:

- Number rows 0 through 3.
- Number columns 5 through 14.
- Use the hexadecimal numbering system for numbering rows and columns.
- Put two blanks between each column.
- Display 5 columns in each group.
- Start printing in the fourth column from the left margin.
- View all non-reserved fields.
- Print the field label ARRENTRY.

One way to code the macro:

```assembly
BLSQMFLD NAME=ARRAYX,ARRAY=((5,14),(0,3)),VIEW=X'0200', X
  STRTCOL=3,COLSEP=2,COLNUM=5,NOLABEL,ORDER=(2,1)
BLSQMFLD NAME=ARRENTRY,OFF=0,LEN=4,ARRAY=END,VIEW=X'0200'
```
Chapter 14. BLSQSHDR — Generate model subheader

Description

The BLSQSHDR macro defines a text string, called a subheader, and makes it appear as part of the output of the model processor. Subheaders are also used to contain the character string or strings to be compared with the contents of a field within the data being formatted, and to determine whether the data is to be treated as hexadecimal or EBCDIC. See the description of the DTYPE=QANY parameter on the BLSQMFLD macro.

BLSQSHDR, with its text string, should be placed after the end of the format model definition. Create a format model definition by coding two BLSQMDEF macros, one at the beginning and one at the end of the definition. The BLSQMFLD macros define the data fields of the format model. They are included between the two BLSQMDEF macros. The SHDR fields of the BLSQMFLD macros refer to text strings (subheaders) that the user places after the end of the model definition. This is the order of the macros:

```
BLSQMDEF
BLSQMFLD  .  .  .
BLSQMFLD
BLSQMDEF
BLSQSHDR
```

Each BLSQSHDR macro placed after the end of the model must have a label that the BLSQMFLD macros within the model can reference. The text string of the BLSQSHDR macro is enclosed in single quotation marks. L(x) can also be coded when the length of the string is different than the length of the enclosed text string.

See [z/OS MVS IPCS Customization](https://www.ibm.com) for information about format models.

Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.
**BLSQSHDR macro**

**Syntax**

This is the standard form of the BLSQSHDR macro:

\[
\text{name name} \\
\text{/bslash} \\
\text{BLSQSHDR} \\
\text{b} \\
\text{L(x)} \\
\text{‘text’}
\]

- **name**: Symbol. Begin name in column 1.
- **b**: One or more blanks must precede BLSQSHDR.
- **BLSQSHDR**: One or more blanks must follow BLSQSHDR.
- **L(x)**: \( x \): Length of subheader - when other than length of actual text
- **‘text’**: text: Text of subheader

**Parameters**

The parameters are explained as follows:

\[
\text{L(x)} \\
\text{specifies the length of the subheader. Specify the length only when it is different from the length of the enclosed text string.}
\]

**Examples**

\[
\text{SHDR01 BLSQSHDR ‘This is a subheader’} \\
\text{SHDR02 BLSQSHRD L(6) ‘ ’}
\]
Chapter 15. BLSRDRPX — Map dump record prefix

Description

The BLSRDRPX macro creates a map of the dump record prefix. The dump record prefix contains the title of the dump and other information needed for interpretation of the dump.


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

```
name name
  name: Symbol. name must begin in column 1 and it cannot exceed four characters in length.
b
  One or more blanks must precede BLSRDRPX.
BLSRDRPX
b
  One or more blanks must follow BLSRDRPX.
```

```
ABITS=31 Default: ABITS=64
ABITS=64
DSECT=YES Default: DSECT=YES
DSECT=NO
```
BLSRDRPX macro

Parameters

The parameters are explained as follows:

**ABITS=31**
**ABITS=64**
   Specifies whether 31-bit or 64-bit storage is to be generated.

**DSECT=YES**
**DSECT=NO**
   Generates a DSECT for the dump record prefix (DSECT=YES) or generates an initialized set of DCs for the dump record prefix (DSECT=NO).
Chapter 16. BLSRESSY — Map IPCS symbol table data

Description

BLSRESSY maps a structure that is part of the interface between an interactive problem control system (IPCS) exit service and an IPCS exit routine. See z/OS MVS IPCS Customization for information about IPCS exit services. See z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a mapping of the BLSRESSY data area.

Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

This is the standard form of the BLSRESSY macro:

\[
\text{name} \\
\text{name: Symbol. Begin name in column 1.} \\
\text{b} \\
\text{One or more blanks must precede BLSRESSY.} \\
\text{BLSRESSY} \\
\text{b} \\
\text{One or more blanks must follow BLSRESSY.} \\
\text{DSECT=YES} \\
\text{DSECT=NO} \\
\text{Default: DSECT=YES} \\
\text{ABITS=31} \\
\text{ABITS=64} \\
\text{Default: ABITS=31}
\]
BLSRESSY macro

Note: Users must supply a label (name), and start it in column 1 of the BLSRESSY macro. When the BLSRESSY macro is executed, the label becomes the record name and the prefix to the name of each field in the record.

Parameters

The parameters are explained as follows:

DSECT=YES
DSECT=NO
  Specifies whether the record mapped by BLSRESSY is to be mapped as a DSECT (YES) or not (NO).

ABITS=31
ABITS=64
  Specifies whether the record mapped by BLSRESSY is to be mapped as 31-bit or 64-bit.

Example

Map the IPCS symbol table record but not as a DSECT.

ESSY BLSRESSY DSECT=NO
Chapter 17. BLSRNAMP — Map the name service parameter list

Description

BLSRNAMP maps the name service parameter list. Use this parameter list when calling the name service from within an installation-written interactive problem control system (IPCS) exit routine.

The name service is used to convert an STOKEN or real address of a data space ASTE into:

- An ASID for an address space
- A data space or hiperspace name and owning ASID
- A common data space (CADS)


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

```
name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede BLSRNAMP.

BLSRNAMP

b

One or more blanks must follow BLSRNAMP.
```
BLSRNAMP macro

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSECT=YES</td>
<td>Default: DSECT=YES</td>
</tr>
<tr>
<td>DSECT=NO</td>
<td></td>
</tr>
</tbody>
</table>

Note: Users must supply a label *(name)*, and start it in column 1 of the BLSRNAMP macro. When the BLSRNAMP macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

Parameters

The parameters are explained as follows:

- **DSECT=YES**
- **DSECT=NO**

A name service parameter list is mapped with a DSECT *(DSECT=YES)* or a name service parameter list is mapped, but no DSECT is generated *(DSECT=NO)*.

Example

Code the following to map the name service parameter list, but not as a DSECT. All fields will appear with the prefix NAMP.

```
NAMP   BLSRNAMP DSECT=NO
```
Chapter 18. BLSRPRD — Map dump record

Description

The BLSRPRD macro creates a map of the dump record.

See [z/OS MVS IPCS Customization](http://www.ibm.com/systems/z/os/zos/bkserv/) for information about formatting dump data.


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

This is the standard form of the BLSRPRD macro:

```
name name: Symbol. name must begin in column 1 and it cannot exceed four characters in length.

b One or more blanks must precede BLSRPRD.

BLSRPRD

b One or more blanks must follow BLSRPRD.

DSECT=YES

Default: DSECT=YES
DSECT=NO
```
BLSRPRD macro

Parameters

The parameters are explained as follows:

DSECT=YES
DSECT=NO

Generates a DSECT for the dump record (DSECT=YES) or generates an initialized set of DCs for the dump record (DSECT=NO).
Chapter 19. BLSRPWHS — Map the WHERE service parameter list

Description

BLSRPWHS maps the WHERE service parameter list. Use this parameter list when calling the WHERE service from within an installation-written interactive problem control system (IPCS) exit routine.

The WHERE service fills in the WHERE service parameter list with information describing the system area in which the passed address resides.


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

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

b

BLSRPWHS

b
```

Default: DSECT=YES
BLSRPWHS macro

ABITS=31
ABITS=64

**Default**: ABITS=31

**Note**: Users must supply a label (*name*), and start it in column 1 of the BLSRPWHS macro. When the BLSRPWHS macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

**Parameters**

The parameters are explained as follows:

- **DSECT=**
  - **YES**
  - **NO**

  A WHERE service parameter list is mapped with a DSECT (DSECT=YES) or a WHERE service parameter list is mapped, but no DSECT is generated (DSECT=NO).

- **ABITS=**
  - **31**
  - **64**

  Either a 31-bit or a 64-bit storage map is to be referenced.

**Example**

Code the following to map the WHERE service parameter list, but not as a DSECT. All fields will appear with the prefix PWHS.

```
PWHS BLSRPWHS DSECT=NO
```
Chapter 20. BLSRSASY — Map IPCS storage map data

Description

BLSRSASY maps a structure that is part of the interface between an interactive problem control system (IPCS) exit service and an IPCS exit routine. See z/OS MVS IPCS Customization for information about IPCS exit services. See z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a mapping of the BLSRSASY data area.

Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

```
name name
DSECT=YES
DSECT=NO
ABITS=31
ABITS=64
```

name: Symbol. Begin name in column 1.

b

One or more blanks must precede BLSRSASY.

BLSRSASY

b

One or more blanks must follow BLSRSASY.
BLSRSASY macro

**Note:** Users must supply a label (*name*), and start it in column 1 of the BLSRSASY macro. When the BLSRSASY macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

**Parameters**

The parameters are explained as follows:

**DSECT=**

- **DSECT=YES**
  - An SA record is mapped with a DSECT (DSECT=YES) or an SA record is mapped, but no DSECT is generated (DSECT=NO).

- **DSECT=NO**

**ABITS=**

- **ABITS=31**
  - A structure is returned containing either 31-bit or 64-bit fields.

- **ABITS=64**

**Example**

Code the following to map an SA record, but not as a DSECT. All fields will appear with the prefix SASY.

```
SASY   BLSRSASY DSECT=NO
```
Chapter 21. BLSRXMSP — Map the storage map service parameter list

Description

BLSRXMSP maps the storage map service parameter list. Use this parameter list when calling the storage map service from within an installation-written interactive problem control system (IPCS) exit routine.

The storage map service allows exit routines to process storage map entries and to obtain data represented by the storage map entries.


Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

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

DSECT=YES

Default: DSECT=YES

DSECT=NO
Note: Users must supply a label (name), and start it in column 1 of the BLSRXMSP macro. When the BLSRXMSP macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

Parameters

The parameters are explained as follows:

DSECT=YES
DSECT=NO

A storage map service parameter list is mapped with a DSECT (DSECT=YES) or a storage map service parameter list is mapped, but no DSECT is generated (DSECT=NO).

Example

Code the following to map the storage map service parameter list, but not as a DSECT. All fields will appear with the prefix XMSP.

XMSP BLSRXMSP DSECT=NO
Chapter 22. BLSRXSSP — Map the symbol service parameter list

Description

BLSRXSSP maps the symbol service parameter list. Use this parameter list when calling the symbol service from within an installation-written interactive problem control system (IPCS) exit routine.

The symbol service enables exit routines to process symbols and obtain data represented by the symbols.

See [z/OS MVS IPCS Customization](http://www.ibm.com/systems/z/os/zos/bkserv/) for information about the symbol service. See [z/OS MVS Data Areas in z/OS Internet Library at](http://www.ibm.com/systems/z/os/zos/bkserv/) for a mapping of the BLSRXSSP data area.

Environment

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

Programming requirements

None.

Restrictions

None.

Register information

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

Performance implications

None.

Syntax

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

```
name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede BLSRXSSP.

BLSRXSSP

b

One or more blanks must follow BLSRXSSP.
```

DSECT=YES

Default: DSECT=YES

DSECT=NO
BLSRXSSP macro

**Note:** Users must supply a label (*name*), and start it in column 1 of the BLSRXSSP macro. When the BLSRXSSP macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

**Parameters**

The parameters are explained as follows:

- **DSECT=YES**
- **DSECT=NO**

A symbol service parameter list is mapped with a DSECT (DSECT=YES) or a symbol service parameter list is mapped, but no DSECT is generated (DSECT=NO).

**Example**

Code the following to map the symbol service parameter list, but not as a DSECT. All fields will appear with the prefix XSSP.

```
XSSP  BLSRXSSP  DSECT=NO
```
Chapter 23. BLSUPPR2 — Map the expanded print service parameter list

Description

BLSUPPR2 maps the expanded print service parameter list. Use this parameter list when calling the expanded print service from within an installation-written interactive problem control system (IPCS) exit routine.

The expanded print service provides a means for exit routines to write data to both the terminal and the IPCS print file.


Environment

Because BLSUPPR2 is not an executable macro, there are no specific environment requirements.

Programming requirements

None.

Restrictions

None.

Register information

Because BLSUPPR2 is not an executable macro, there is no need to save and restore register contents.

Performance implications

None.

Syntax

The standard form of the BLSUPPR2 macro is written as follows:

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

b: One or more blanks must precede BLSUPPR2.

BLSUPPR2

b: One or more blanks must follow BLSUPPR2.

DSECT=YES

Default: DSECT=YES
```
BLSUPPR2 macro

DSECT=NO

---

**Note:** Users must supply a label (*name*), and start it in column 1 of the BLSUPPR2 macro. When the BLSUPPR2 macro is processed, the label becomes the record name and the prefix to the name of each field in the record.

**Parameters**

The parameters are explained as follows:

- **DSECT=YES**
- **DSECT=NO**

An expanded print parameter list is mapped with a DSECT (DSECT=YES) or an expanded print parameter list is mapped, but no DSECT is generated (DSECT=NO).

**Example**

Code the following to map the expanded print service parameter list, but not as a DSECT. All fields will appear with the prefix PPR2.

```
PPR2 BLSUPPR2 DSECT=NO
```
Chapter 24. CALL — Pass control to a control section

Description

The CALL macro passes control to a control section at a specified entry point as follows:

- OVERLAY: The overlay segment containing the designated entry point is brought into virtual storage if required, and control is passed to the segment. Refer to [z/OS MVS Program Management: User's Guide and Reference][1] and [z/OS MVS Program Management: Advanced Facilities][2] for details on overlay.

- NONOVERLAY: If a symbol is designated, the linkage editor includes the load module containing that entry point in the same load module containing the CALL instruction. When the CALL macro is executed, control is passed to the control section at the specified entry point.

The linkage relationship established when control is passed is the same as that created by a BAL instruction; that is, the issuing program expects control to be returned. The control program is not involved in passing control, so the reusability of the called program must be ensured by the user.

An address parameter list can be constructed and a calling sequence identifier can be provided.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state and any PSW key
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: PASN=HASN=SASN or PASN¬=SASN¬=HASN
- **AMODE**: 24- or 31- or 64- bit
- **ASC mode**: Primary or Access register (AR)
- **Interrupt status**: No requirement
- **Locks**: No requirement
- **Control parameters**: Must be in the caller's primary address space or be in an address or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is to execute in 31-bit addressing mode, you must use the MVS/SP Version 2 macro expansion or a later version. You cannot use the CALL macro to pass control to a program in a different addressing mode.

AR mode programs and primary mode programs can invoke the CALL macro. Before an AR mode program invokes this macro, the program must issue SYSSTATE ASCENV=AR to tell the CALL macro to generate code that is appropriate for AR mode. Before a 64-bit Amode program invokes this macro, the program must issue SYSSTATE AMODE64=YES to tell the CALL macro to generate code that is appropriate for Amode 64.

**IBM recommends** that you do not use asynchronous exit routines in an overlay program. If you choose to do so, you must ensure that:

[1]: z/OS MVS Program Management: User's Guide and Reference
[2]: z/OS MVS Program Management: Advanced Facilities
The overlay segment containing the asynchronous exit routine is already in storage at the time the system invokes the routine, and this segment will not be overlaid by another segment during the routine’s execution.

If the asynchronous exit routine calls a routine in an overlay segment, that segment is already in storage and will not be overlaid by another segment during the called routine’s execution.

Register information

On entry to the called program, the register contents are as follows:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Address of the parameter list, if present</td>
</tr>
<tr>
<td>14</td>
<td>Return address</td>
</tr>
<tr>
<td>15</td>
<td>Entry address of the called program</td>
</tr>
</tbody>
</table>

Syntax

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

```
name name

, (addr)
, (addr),VL

,PLIST4=YES
,PLIST4=NO

,PLIST8=YES
,PLIST8=NO

,PLIST8ARALET=NO
,PLIST8ARALET=YES

,ID=id nmbr

,LINKINST=instruction
```

- `name`: Symbol. Begin `name` in column 1.
- `b`: One or more blanks must precede CALL.
- `CALL`: One or more blanks must follow CALL.
- `entry name`: Symbol or register (15).
- `addr`: A-type address, or register (2) - (12).
- `Note`: `addr` is one or more addresses, separated by commas. For example, `(addr,addr,addr)`
- `Default`: None.
Parameters

The parameters are explained as follows:

_entry name_

Specifies the entry name to be given control.

,(addr)

Specifies an address or addresses to be passed to the called program. CALL expands each address inline to a fullword boundary and builds a parameter list with the addresses in the order specified. When the called program receives control, register 1 contains the address of the parameter list. If this parameter is not coded, register 1 is not altered.

When an AR mode caller uses either:

- a parameter list with 4 bytes per entry; or
- a parameter list with 8 bytes per entry and specifies PLIST8ARALETs=YES,

the addresses passed to the subtask are in the first part of the parameter list and their associated ALETS are in the second part. For a non-AR mode caller, or for an AR mode caller using a parameter list with 8 bytes per entry without PLIST8ARALETs=YES, ALETS are not passed in the parameter list. When ALETS are passed in the parameter list, the ALETS occupy consecutive 4-byte fields, whether the parameter list is 4 or 8 bytes per entry. See the description of the PLIST4 and PLIST8 keywords below for more information about controlling the bytes-per-entry in the parameter list. See the description of the PLIST8ARALETs keyword below for more information about ALETS and 8-bytes-per-entry parameter lists. See "User parameters" on page 4 for an example of passing a parameter list in AR mode.

When using a 4-bytes-per-entry parameter list, specify VL=1 when you pass a variable number of parameters. VL=1 results in setting the high-order bit of the last address to 1. The 1 in the high-order bit identifies the last address parameter (which is not the last word in the list when the ALETS are also saved). When using an 8-bytes-per-entry parameter list, VL=1 is not valid.

_Note:_ If you specify only one address for PARAM= and you are not using register notation, you do not need to enter the parentheses.

,PLIST4=YES
,PLIST4=NO

,PLIST8=YES
,PLIST8=NO

Defines the size of the parameter list entries for a parameter list to be built by CALL based on the address or addresses to be passed to the called program.

PLIST4 and PLIST8 cannot be specified together. If neither is specified, the default is:

- If running AMODE 64, PLIST8=YES
- If not running AMODE 64, PLIST4=YES

If running AMODE 64 and PLIST4=YES is specified, the system builds a 4-bytes-per-entry parameter list just as it would if the program were running AMODE 24 or AMODE 31 and did not specify PLIST4 or PLIST8.

If running AMODE 24 or AMODE 31 and PLIST8 is specified, the system builds an 8-bytes-per-entry parameter list just as it would if the program were running AMODE 64 and did not specify PLIST4 or PLIST8.
CALL macro

,PLIST8ARALETs=NO
,PLIST8ARALETs=YES
If there is to be an 8-byte-per-entry parameter list and the invoker is in AR mode, indicates if the parameter list is also to contain the ALETs associated with the addresses. Otherwise, this parameter is ignored.

,PLIST8ARALETs=NO
Indicates that the 8-byte-per-entry parameter list is to consist of just the 8-byte addresses.

,PLIST8ARALETs=YES
Indicates that the 8-byte-per-entry parameter list is to consist of the following two parts:
• All the 8-byte addresses,
• All the associated ALETs in consecutive 4-byte fields.

,ID=ID nmbr
Specifies a 2-byte identifier useful for debugging purposes only. The last fullword of the macro expansion is a NOP instruction containing the identifier value in bytes 3 and 4.

,LINKINST=instruction
Specifies the linkage instruction to use in this macro. The default is LINKINST=BALR.

Return and reason codes
The CALL macro does not generate any return codes. A return code in GPR 15 or AR 15 is placed there by the called program.

Example
Call the entry point contained in register 15, and pass three addresses to the control program.
CALL (15),(ADDR1,ADDR2,ADDR3)

CALL-List form
Use the list form of the CALL macro to construct a nonexecutable problem program parameter list. This list form generates only ADCONs of the address parameters. You can refer to this problem program parameter list in the execute form of a CALL, LINK, LINKX, ATTACH, ATTACHX, XCTL, or XCTLX macro.

Syntax
The list form of the CALL macro is written as follows:

```
  name

  b
  CALL
  b
```

name: Symbol. Begin name in column 1.

One or more blanks must precede CALL.

One or more blanks must follow CALL.
CALL macro

.addr: A-type address.

Note: .addr is one or more addresses, separated by commas. For example, (addr, addr, addr)

Default: PLIST8ARALETS=NO

Parameters

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

,MF=L

Specifies the list form of the CALL macro.

CALL-Execute form

The execute form of the CALL macro can refer to and modify a remote problem program parameter list. Only executable instructions and a VCON of the entry point are generated.

Syntax

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

```
name

b

CALL

b
```

```
entry name

,(addr)
,VL

,PLIST8ARALETS=NO
,PLIST8ARALETS=YES

,ID=id nmbr

,LINKINST=instruction

,MF=(E, prob addr)
```

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

One or more blanks must precede CALL.

One or more blanks must follow CALL.

entry name: Symbol or register (15).

.addr: RX-type address, or register (2) - (12).

Note: .addr is one or more addresses, separated by commas. For example, (addr, addr, addr)

Default: PLIST8ARALETS=NO

id nmbr: Symbol or decimal digit, with a maximum value of 4095.

Default: LINKINST=BALR

prob addr: RX-type address, or register (1) or (2) - (12).```
CALL macro

Parameters

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

\[ \text{MF} = \{E, \text{prob} \, \text{addr}\} \]

Specifies the execute form of the CALL macro. This form uses a remote problem program parameter list. If the address parameters are also specified in this form, the ADCONs of the parameter are placed on contiguous fullword boundaries beginning at the address specified in the MF parameter, and sequentially overlaying corresponding fullwords in the existing list.
Chapter 25. CHAP — Change dispatching priority

Description

CHAP changes the dispatching priority of the task or any of its subtasks relative to the other tasks in the address space. It does not change the priority relative to other tasks in the system. CHAP may also change the limit priority of a subtask. (See the topic “Priorities” in the z/OS MVS Programming: Assembler Services Guide) The algebraic sum of the priority value and the dispatching priority of the subject task determine the new dispatching priority.

- If the subject task is the task executing CHAP, its dispatching priority is set equal to the sum of the priority value and the dispatching priority. This value is not set to less than zero or greater than the limit priority for the task. Its limit priority is unaffected.
- If the subject task is a subtask of the task executing CHAP, its dispatching priority is set equal to the sum of the priority value and the dispatching priority. This value is not set to less than zero or greater than the limit priority of the task executing CHAP. After this modification, if the subtask’s dispatching priority exceeds its limit priority, the limit priority is made equal to the dispatching priority.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state and any PSW key
- **Dispatchable unit mode**: Task
- **Cross memory mode**: PASN=HASN=SASN
- **AMODE**: 24- or 31- or 64-bit
- **ASC mode**: Primary
- **Interrupt status**: Enabled for I/O and external interrupts
- **Locks**: No lock held
- **Control parameters**: Must be in the primary address space.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the CHAP 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:
### CHAP macro

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

#### Performance implications

None.

#### Syntax

The CHAP macro is written as follows:

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

b         One or more blanks must precede CHAP.

CHAP

b         One or more blanks must follow CHAP.

priority value

\slash    priority value: Symbol, decimal digit, or register (0) or (2) - (12).

,'S'
\slash    \texttt{tcb addr}: RX-type address, or register (1) or (2) - (12).
\slash    \textbf{Default}: ‘S’
\slash    \texttt{,tcb addr}

,RELATED=value

\slash    value: Any valid macro keyword specification.
```

#### Parameters

The parameters are explained as follows:

- **priority value**
  - Specifies the signed value to be added to the dispatching priority of the specified task. If the value is negative and contained in a register, it must be in two's complement form.

- **,‘S’**
  - \texttt{tcb addr}: RX-type address, or register (1) or (2) - (12).
  - \textbf{Default}: ‘S’

- **,RELATED=value**
  - \textit{value}: Any valid macro keyword specification.

**Note:** TCB must reside in 24-bit addressable storage.

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

The RELATED parameter is available on macros that provide opposite services (for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE) and on macros that relate to previous occurrences of the same macros (for example, CHAP and ESTAE).

You may use the RELATED parameter as follows:

CHAPUP CHAP 1,'S',RELATED=(CHAPDOWN,'UP PRIORITY')

CHAPDOWN CHAP -1,'S',RELATED=(CHAPUP 'RESUME INITIAL PRIORITY')

ABEND codes

07F
12C
22C

See [z/OS MVS System Codes](#) for an explanation and programmer responses for these codes.

Return and reason codes

None.

Example 1

Lower the dispatching priority of the subtask TCB by two. The subtask TCB's address is in a fullword which register 1 addresses. The subtask TCB will be repositioned on the dispatching queue in accordance with its new dispatching priority.

CHAP  -2,(1)

Example 2

Reposition the TCB of the task issuing CHAP at the bottom of the group of TCBs on the dispatching queue for the address space, having the same dispatching priority as that task.

CHAP 0
CHAP macro
Chapter 26. CnzConv -- Convert console name and ID

Description

Application programmers can retrieve information about an input console name or console ID using the CnzConv macro.

You can use the CnzConv service to obtain the following information:

- The console name associated with an input console ID.
- The console ID associated with an input console name.
- The console status (active or inactive).
- The console type (MCS, SMCS, Subsystem, EMCS, or Special).
- The console subtype (a value of Internal, Instream, or Unknown, for a console type of Special).
- The system name where a console is active.
- The logic unit name of an SMCS console.
- The subsystem owner name of a subsystem console.
- The subsystem ASID of a subsystem console.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state and any PSW key
- **Dispatchable unit mode**: Task
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 31- or 64-bit
- **ASC mode**: Primary or access register (AR)
- **Interrupt status**: Enabled for I/O and external interrupts
- **Locks**: No locks held
- **Control parameters**: Must be in the primary address space.

Programming requirements

Before issuing CnzConv, take the following actions:

- If in access register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
- If in access register ASC mode, the ALET associated with the CnzConv parameter list must be zero.
- If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
- If a list and execute form of CnzConv is being used, clear the list form of the parameter list before each use.

Programming considerations

Make sure you have all the information necessary to process the results of your CnzConv query. For example, SUBSYSASID will only be returned for an active subsystem console. As a result on your CnzConv query, you should request the CONSOLESTATUS, CONSOLETEYPE, and SUBSYSASID at a minimum. Then after verifying the console status is active and the console type is subsystem, it can be concluded that the SUBSYSASID has been returned.

Restrictions

There are six reserved console names. Do not use any of the following as console names:
CnzConv macro

- HC
- LOGON
- LOGOFF
- OPERLOG
- SYSLOG
- UNKNOWN

Input register information
Before issuing the CnzConv 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 CnzConv macro is written as follows:

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

b             One or more blanks must precede CnzConv.

CnzConv

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

InConsoleName = InConsoleName Addr

`InConsoleName Addr`: RX-type address or register (2) - (12).

InConsoleId = InConsoleId Addr

`InConsoleId Addr`: RX-type address or register (2) - (12).
CnzConv macro

\texttt{.OutConsoleId = OutConsoleId Addr} \quad \text{OutConsoleId Addr: RX-type address or register (2)-(12).}

\texttt{.OutConsoleName = OutConsoleName Addr} \quad \text{OutConsoleName Addr: RX-type address or register (2)-(12).}

\texttt{.ConsoleStatus = ConsoleStatus Addr} \quad \text{ConsoleStatus Addr: RX-type address or register (2)-(12).}

\texttt{.ConsoleType = ConsoleType Addr} \quad \text{ConsoleType Addr: RX-type address or register (2)-(12).}

\texttt{.ConsoleSubType = ConsoleSubType Addr} \quad \text{ConsoleSubType Addr: RX-type address or register (2)-(12).}

\texttt{.Sysname = Sysname Addr} \quad \text{Sysname Addr: RX-type address or register (2)-(12).}

\texttt{.SMCS_LU = SMCS_LU Addr} \quad \text{SMCS_LU Addr: RX-type address or register (2)-(12).}

\texttt{.SubsysOwnerName = SubsysOwnerName Addr} \quad \text{SubsysOwnerName Addr: RX-type address or register (2)-(12).}

\texttt{.SubsysASID = SubsysASID Addr} \quad \text{SubsysASID Addr: RX-type address or register (2)-(12).}

\texttt{.RtnCode = RtnCode Addr} \quad \text{RtnCode Addr: RX-type address or register (2)-(12).}

\texttt{.RsnCode = RsnCode Addr} \quad \text{RsnCode Addr: RX-type address or register (2)-(12).}

---

**Parameters**

The parameters are explained as follows:

**InConsoleName**

Belongs to a set of mutually exclusive keys. It is the name (RS-type), or address in register (2)-(12), of an 8 character input that is the input console name to query.

**InConsoleId**

Belongs to a set of mutually exclusive keys. It is the name (RS-type), or address in register (2)-(12), of a fullword input that is the input console ID to query.

**OutConsoleId**

Is the name (RS-type), or address in register (2)-(12), of an optional fullword output that contains the console ID of the requested console when the query completes successfully. When the query does not complete successfully, OutConsoleId will contain binary zeros.

**OutConsoleName**

Is the name (RS-type), or address in register (2)-(12), of an optional 8 character output that contains the console name of the requested console when the query completes successfully. When the query does not complete successfully, OutConsoleName will contain binary zeros.

**ConsoleStatus**

Is the name (RS-type), or address in register (2)-(12), of an optional byte output that contains the status of the requested console when the query completes successfully. When the query does not complete successfully, ConsoleStatus
will contain binary zeros which means not applicable. A console status will be returned for a console type of MCS, SMCS, Subsys, or EMCS. A console status of binary zeros, which means not applicable, will be returned for a console type of Special. The console statuses are: active, inactive. The constants for the console statuses are defined in macro IEZVG200.

**ConsoleType**

Is the name (RS-type), or address in register (2)-(12), of an optional byte output that contains the type of the requested console when the query completes successfully. When the query does not complete successfully, ConsoleType will contain binary zeros. Any query that completes successfully will have a console type set. The console types are: MCS, SMCS, Subsys, EMCS, Special. The constants for the console types are defined in macro IEZVG200.

**ConsoleSubType**

Is the name (RS-type), or address in register (2)-(12), of an optional byte output that contains the subtype of the requested console when the query completes successfully. When the query does not complete successfully, or completes successfully but the requested console does not have a console subtype, ConsoleSubtype will contain binary zeros which means not applicable. Currently, console subtypes are only returned for a console type of Special. The console subtypes are: Internal, Instream, Unknown, JES3. The constants for the console subtypes are defined in macro IEZVG200.

**Sysname**

Is the name (RS-type), or address in register (2)-(12), of an optional 8 character output that contains the system name where the requested console is active when it has a console status of active, and a console type of MCS, SMCS, Subsys, or EMCS, and the query completes successfully. When the console status is not active, or the console type is Special, or the query does not complete successfully, Sysname will contain binary zeros.

**SMCS_LU**

Is the name (RS-type), or address in register (2)-(12), of an optional 8 character output that contains the LU name of the requested console when it has a console type of SMCS and the query completes successfully. When the console type is not SMCS or the query does not complete successfully, SMCS_LU will contain binary zeros.

**SubsysOwnerName**

Is the name (RS-type), or address in register (2)-(12), of an optional 8 character output that contains the subsystem owner name of the requested console when it has a console type of Subsys, and a console status of active, and the query completes successfully. When the console type is not Subsys, or the console status is not active, or the query does not complete successfully, SubsysOwnerName will contain binary zeros.

**SubsysASID**

Is the name (RS-type), or address in register (2)-(12), of an optional halfword output that contains the ASID of the requested console when it has a console type of Subsys, and a console status of active, and the query completes successfully. When the console type is not Subsys, or the console status is not active, or the query does not complete successfully, SubsysASID will contain binary zeros.

**RtnCode**

Is the name (RS-type) of an optional fullword output variable, or register (2)-(12), into which the return code is to be copied from GPR 15.
CnzConv macro

RsnCode
Is the name (RS-type) of an optional fullword output variable, or register (2)-(12), into which the reason code is to be copied from GPR 0.

ABEND codes
077

Return and reason codes
When the CnzConv macro returns control to your program, if any return code is specified, it will be copied from Register 15 into RtnCode, if any reason code is specified, it will be copied from Register 0 into RsnCode. The following table shows the meanings and actions for the hexadecimal return codes and reason codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>N/A</td>
<td>Name: CnzConvRc0_Ok</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The input console name or ID was found and the applicable requested data was returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>4</td>
<td>xxxxxxxx</td>
<td>Name: CnzConvRc4_ConditionallyOK</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The request completed successfully with an exception.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Examine the reason code to determine how to proceed.</td>
</tr>
<tr>
<td>4</td>
<td>xxxx0401</td>
<td>Name: CnzConvRsn401_IdNotFound</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The console ID in InConsoleId is not associated with any console.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the console ID in InConsoleId to be the ID of a defined console or take appropriate action when the console ID in InConsoleId was not found.</td>
</tr>
<tr>
<td>4</td>
<td>xxxx0402</td>
<td>Name: CnzConvRsn402_NameNotFound</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The console name in InConsoleName is not associated with any console.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the console name in InConsoleName to be the name of a defined console or take appropriate action when the console name in InConsoleName was not found.</td>
</tr>
<tr>
<td>4</td>
<td>xxxx0403</td>
<td>Name: CnzConvRsn403_NameIsReserved</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> The input console name is a reserved console name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the console name in InConsoleName to be the name of a defined console or take appropriate action when the console name in InConsoleName is reserved.</td>
</tr>
<tr>
<td>8</td>
<td>xxxxxxxx</td>
<td>Name: CnzConvRc8_SpecificationError</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Meaning:</strong> An error was detected in the CnzConv parameter list. None of the requested data has been returned.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Correct the CnzConv parameter list. Examine the reason code to determine how to proceed.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Reason Code</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 8 xxxx0801  | xxxx0801    | Name: CnzConvRsn801_BadPlistVer  
  Meaning: The PLISTVER in the CnzConv parameter list is incorrect.  
  Action: Correct the PLISTVER in the CnzConv parameter list. |
| 8 xxxx0802  | xxxx0802    | Name: CnzConvRsn802_ExtraneousInput  
  Meaning: InConsoleName and InConsoleId are mutually exclusive keywords but both were specified.  
  Action: Specify one and only one of the following keywords: InConsoleName or InConsoleId. |
| 8 xxxx0803  | xxxx0803    | Name: CnzConvRsn803_IncompleteArgs  
  Meaning: Neither InConsoleName nor InConsoleId keyword was specified.  
  Action: Specify one and only one of the following keywords: InConsoleName or InConsoleId. |
| 8 xxxx0804  | xxxx0804    | Name: CnzConvRsn804_NameInvalidSyntax  
  Meaning: The console name in InConsoleName is syntactically invalid and cannot be a console name.  
  Action: Correct the input console name. |
| 8 xxxx0805  | xxxx0805    | Name: CnzConvRsn805_RsvSpaceNotZero  
  Meaning: The reserved space in the CnzConv parameter list is not binary zeros.  
  Action: Correct the CnzConv parameter list so that the reserved space contains binary zeros. |
| C xxxxxxxxx | xxxxxxxxx   | Name: CnzConvRcC_Error  
  Meaning: The request failed to complete successfully. None of the requested data has been returned.  
  Action: Examine the reason code to determine how to proceed. |
| C xxxx0C01  | xxxx0C01    | Name: CnzConvRsnC01_NotAvailable  
  Meaning: The CnzConv service is not available at this time. This typically would not occur after system initialization.  
  Action: Resubmit your request at a later time. |
| C xxxx0C02  | xxxx0C02    | Name: CnzConvRsnC02_IncorrectEnv  
  Meaning: The CnzConv service was invoked in an incorrect environment.  
  Action: Invoke the CnzConv service in the correct environment. |
### CnzConv macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 10          | xxxxxxxx    | Name: CnzConvRc10_UnexpectedError  
  **Meaning:** Unexpected failure occurred. The outcome of the request is unpredictable, meaning that it might have completed successfully, or partially, or not at all. All, some, or none of the data requested has been returned. A dump might have been taken.  
  **Action:** Examine the reason code to determine how to proceed. |
| 10          | xxxx1001    | Name: CnzConvRsn1001_SevereError  
  **Meaning:** The CnzConv service was unable to complete your request due to an unexpected error processing the CnzConv request.  
  **Action:** Supply the return code, reason code, and the dump to the appropriate IBM support personnel. |

### Example

A typical application of CnzConv would be in an MPF message exit. A message exit could be built to reroute a message to an active console in your system or sysplex. As shown in the following example, you can change the routing information or the message destination depending on the status of the console:

- If CnzConv indicates the console is active and you want to send the message only to that active console, add the necessary code at the CONTINUE label.
- If CnzConv indicates the console is inactive and you want to route this message to an active console, add the necessary code to the NOTACTIVE label.

This example assumes that you would have function specified at the labels referenced as locations of branch instructions (block comments are also in the example showing where these would be).

NYC8CONV CSECT  
NYC8CONV AMODE 31  
NYC8CONV RMODE ARY  
******************************************************************************  
* REGISTER ASSIGNMENTS  
  *  
  *  
******************************************************************************  
REG0 EQU 0  
REG1 EQU 1  
REG2 EQU 2  
REG3 EQU 3  
REG4 EQU 4  
REG5 EQU 5  
REG6 EQU 6  
REG7 EQU 7  
REG8 EQU 8  
REG9 EQU 9  
REG10 EQU 10  
REG11 EQU 11  
REG12 EQU 12  
REG13 EQU 13  
REG14 EQU 14  
REG15 EQU 15
CnzConv macro

SPACE 1

**************************************************************
* * STANDARD ENTRY LINKAGE * *
**************************************************************
BAKR REG14,0 SAVE REGS
BALR REG12,0 BASE REG
USING *,REG12 ADDRESSABILITY
MODID ,

**************************************************************
* * OBTAIN DYNAMIC AREA STORAGE. *
**************************************************************

SPACE 1
LA REG0,DYNL LENGTH OF DATA AREAS
GETMAIN RU,LV=(REG0),SP=0 OBTAIN DYNAMIC STORAGE
LR REG11,REG1 ADDRESS RETURNED IN REG1
USING DYNMODEL,REG11 ADDRESSABILITY TO DYNAMIC
XC MYCNZCONV,MYCNZCONV CLEAR CNZCONV PARMLIST

**************************************************************
* * INVOKE CNZCONV FOR A CONSOLE NAME. *
**************************************************************

CnzConv INCONSOLENAME=MYCONSOLENAME, X
OUTCONSOLEID=OUTCONSOLEID, X
OUTCONSOLENAME=OUTCONSOLENAME, X
CONSOLESTATUS=OUTCONSOLESTATUS, X
CONSOLETEYPE=OUTCONSOLETEYPE, X
CONSOLETESUBTYPE=OUTCONSOLETESUBTYPE, X
SYSNAME=OUTSYSNAME, X
SMCS_LU=OUTSMCS_LU, X
SUBSYSOWNERNAME=OUTSUBSYSOWNERNAME, X
SUBSYSASID=OUTSUBSYSASID, X
RTNCODE=CNZCONVRETURNCODE, X
RSNCODE=CNZCONVREASONCODE, X
MF=(E,MYCNZCONV)
CLI CNZCONVRETURNCODE,CNZCONVRC0_OK X
Was the query successful?
BNZ FINISHED No, free storage and return

CLI OUTCONSOLESTATUS,CNZCONV_KSTATUS_INACTIVE X
Was the console inactive?
BE NOTACTIVE Yes, handle not active

CLI OUTCONSOLESTATUS,CNZCONV_KSTATUS_ACTIVE X
Was the console active?
BE CONTINUE

B FINISHED Undefined status, free storage X and return

CONTINUE EQU *

**************************************************************
* * HERE YOU WOULD PROVIDE SUPPORT FOR THE ACTIONS YOU WANTED *
* TO TAKE IF THE CONSOLE WAS ACTIVE. *
**************************************************************

NOTACTIVE EQU *
CnzConv macro

************************************************************
* HERE YOU WOULD PROVIDE SUPPORT FOR THE ACTIONS YOU WANTED
* TO TAKE IF THE CONSOLE WAS INACTIVE.
************************************************************

FINISHED EQU *

FREEMAIN RU, LV=DYNL, A=(REG11), SP=0

PR

MYCONSOLENAME DC 'C'MCSY13EO' CONSOLE NAME

*---------------------------------------------------------------------*
*- DYNAMIC AREA that contains the data returned by CnzConv macro -*
*- plus the CnzConv parameter list -*
*---------------------------------------------------------------------*

DYNMODEL DSECT
DS 0F

OUTCONSOLEID DS F
OUTCONSOLENAME DS CL8
OUTCONSOLESTATUS DS CL1
OUTCONSOLETYPE DS CL1
OUTCONSOLESUBTYPE DS CL1
RSVO000001 DS CL1
OUTSYSNAME DS CL8
OUTSMCS_LU DS CL8
OUTSUBSYSOWNERNAME DS CL8
OUTSUBSYSASID DS H
CNZCONVRETURNCODE DS F
CNZCONVREASONCODE DS F

CNZCONV MF=1,MYCNZCONV) CNZCONV PARAMETER LIST
ORG

DYNL EQU *
*--DYNMODEL DYNAMIC AREA LENGTH
EJECT
IEZVG200 CNZCONV CONSTANTS
EJECT
END

CnzConv -- List form

Use the list form of the CnzConv macro together with the execute form of the macro for programs that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

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

```
  name
  b
  CnzConv
  b
```

*name: Symbol. Begin name in column 1.

* One or more blanks must precede CnzConv.

* One or more blanks must follow CnzConv.
CnzConv macro

\[ .MF=(L, list \, addr) \]
\[ .MF=(L, list \, addr, attr) \]
\[ .MF=(L, list \, addr, 0D) \]

\textit{list \, addr}: Symbol

\textit{attr}: 1- to 60-character input string.

\textit{Default}: 0D

---

Parameters

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

\[ .MF=(L, list \, addr) \]
\[ .MF=(L, list \, addr, attr) \]
\[ .MF=(L, list \, addr, 0D) \]

Specifies the list form of the CnzConv macro. \textit{list \, addr} is the name of a storage area to contain the parameters.

\textit{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 \textit{attr}, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

---

CnzConv -- Execute form

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

\[ name \, \text{Symbol} \]

\begin{align*}
\text{name} & : \text{Symbol. Begin name in column 1.} \\
\text{b} & : \text{One or more blanks must precede CnzConv.} \\
\text{CnzConv} & : \text{One or more blanks must follow CnzConv.} \\
\end{align*}

\begin{align*}
\text{InConsoleName} & = \text{InConsoleName \, Addr} \\
\text{InConsoleId} & = \text{InConsoleId \, Addr} \\
\text{OutConsoleId} & = \text{OutConsoleId \, Addr} \\
\text{OutConsoleName} & = \text{OutConsoleName \, Addr} \\
\text{ConsoleStatus} & = \text{ConsoleStatus \, Addr} \\
\text{ConsoleType} & = \text{ConsoleType \, Addr} \\
\text{ConsoleSubType} & = \text{ConsoleSubType \, Addr} \\
\text{Sysname} & = \text{Sysname \, Addr} \\
\text{SMCS\_LU} & = \text{SMCS\_LU \, Addr} \\
\text{SubsysOwnerName} & = \text{SubsysOwnerName \, Addr} \\
\text{SubsysASID} & = \text{SubsysASID \, Addr} \\
\text{RtnCode} & = \text{RtnCode \, Addr} \\
\text{RsnCode} & = \text{RsnCode \, Addr} \\
\end{align*}

\begin{align*}
\text{InConsoleName \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{InConsoleId \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{OutConsoleId \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{OutConsoleName \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{ConsoleStatus \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{ConsoleType \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{ConsoleSubType \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{Sysname \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{SMCS\_LU \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{SubsysOwnerName \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{SubsysASID \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{RtnCode \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\text{RsnCode \, Addr} & : \text{RX-type address or register (2) - (12).} \\
\end{align*}
CnzConv macro

Default: COMPLETE

Parameters

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

\[ \text{,MF=}(E, \text{list addr, COMPLETE}) \]
\[ \text{,MF=}(E, \text{list addr, NOCHECK}) \]

Specifies the execute form of the CnzConv 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 defaults for omitted optional parameters.

- NOCHECK specifies that the system is not to check for required parameters and is not to supply defaults for omitted optional parameters.
CnzConv macro
Chapter 27. CNZMXURF — UCME look-up service macro

Description

Use the CNZMXURF macro to locate the console control block (UCME) that contains a specific console ID. CNZMXURF must only be used to look up MCS and SMCS console IDs.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state and any PSW key.
- **Dispatchable unit mode**: Task
- **Cross memory mode**: Any PASN, Any HASN, Any SASN
- **AMODE**: 24- or 31-bit
- **ASC mode**: Primary
- **Interrupt status**: Enabled for I/O and external interrupts
- **Locks**: A local lock and the CMS lock must be held.
- **Control parameters**: Not applicable; held in a register.

Programming requirements

Before issuing the CNZMXURF macro, place a 4-byte console ID into a register. No register save area is required; however, the input registers are saved on the linkage stack.

Restrictions

None.

Input register information

Before issuing the CNZMXURF macro, the caller must either place a 4-byte console ID into a register, or directly specify the RX-type address of a 4-byte console ID.

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>UCME pointer</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0–15</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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.
CNZMXURF macro

Performance implications
None.

Syntax
The CNZMXURF macro is written as follows:

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

/bslash
One or more blanks must precede CNZMXURF.

CNZMXURF
/bslash
One or more blanks must follow CNZMXURF.

register: General purpose register Register (2-12) containing a 4-byte console ID.

or
console-ID: RX-type address containing a 4-byte console ID.

,INUSE=NO
Default: INUSE=NO
```

Parameters
The parameters are explained as follows:

```
register
Contains the 4-byte console ID for which the corresponding UCME is to be located.

console-ID
Contains the 4-byte console ID for which the corresponding UCME is to be located.

,INUSE=YES
The UCME corresponding to the input console ID will be returned only if it is initialized and in use.

,INUSE=NO
The UCME corresponding to the input console ID will be returned, even if it is not initialized and not in use.
```

If the system is running in Console Services distributed mode and the console is not active, no UCME address will be returned in register 0 and a return code of 8 will be returned in register 15. For more information about Console Services distributed mode, see z/OS MVS Planning: Operations.

ABEND codes
None.
Return and reason codes

When the CNZMXURF macro returns control to your program, register 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | Meaning: No errors. A pointer to the UCME containing the console ID is returned in register 0.  
Action: None. |
| 04                      | Meaning: A UCME containing the specified console ID was not found. Register 0 contains zero.  
Action: None. |
| 08                      | Meaning: Invalid console ID input or there is no UCME for the specified console ID. Register 0 contains zero.  
• Non-MCS console class was supplied.  
• INTERNAL console ID (0) was supplied.  
• INSTREAM console ID (128) was supplied.  
• UNKNOWN console ID (255) was supplied.  
• There is no UCME for the specified 4-byte console ID or the UCME for the console ID is not initialized and INUSE=YES was specified.  
• If the system is running in Console Services distributed mode and the console is not active, no UCME address will be returned in register 0.  
Action: None. |
| 16                      | Meaning: CNZMXURF service is not available.  
Action: None. |

Example 1
Locate the UCME associated with the 4-byte console ID in register 4.
\[\text{CNZMXURF (4)}\]

Example 2
Locate the UCME associated with the 4-byte console ID stored in field “MYCONID”.
\[\text{CNZMXURF MYCONID}\]
Chapter 28. CNZTRKR — Tracking interface macro

Description

Use the CNZTRKR macro to invoke the tracking facility. This service allows programmers to record events of interest. For more information about the tracking facility, see [z/OS MVS Planning: Operations](#).

Environment

The requirements for the caller are:

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

Programming requirements

Before issuing the CNZTRKR macro, do the following:

- Include the CNZTRPL mapping macro in your program.
- Obtain storage for the CNZTRKR parameter list. TRPL_LEN in CNZTRPL contains the length of the parameter list. The parameter list can be in any type of storage.
- Clear the entire parameter list by setting it to zeros.
- Initialize fields in the parameter list mapped by macro CNZTRPL. You must initialize the following fields:

  - **TRPL_Acro**
  - **TRPL_Version**
  - **TRPL_Track_Info**
  - **TRPL_Track_Data**
  - **TRPL_Violators_Addr**

- The TRPL acronym.
- The current version level of the parameter list. The CNZTRPL mapping macro contains the current version level in TRPL_K_Curr_Version.
- Text that describes the occurrence of this instance. This text can be from 1 to 28 characters in length. Any EBCDIC value is allowed though you should use displayable characters because undisplayable characters may be changed to blanks when displayed on an operator's console or in the hardcopy log. The text cannot be all blank or all hexadecimal zeros.
- Four bytes of data associated with this track instance. Zero is a valid value. The DISPLAY OPDATA operator command will display this value as a hexadecimal number.
- While optional, this field should contain the address of where the event being tracked occurred (perhaps the address to which the service invoking CNZTRKR will return). If set to zero, the tracking facility will attempt to determine the address of the event, but may not be able to determine the exact location. This address is assumed to be a 31-bit address. If a 24-bit address is provided, you must ensure that the high-order byte of the address is zero.
CNZTRKR macro

Restrictions
The caller must not have functional recovery routines (FRRs) established.

Input register information
Before issuing the CNZTRKR 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Syntax
The CNZTRKR macro is written as follows:

```
name name

/bslash

One or more blanks must precede CNZTRKR.

CNZTRKR

/bslash

One or more blanks must follow CNZTRKR.

register register

or

list name

list name: RX-type address.
```

Parameters
The parameters are explained as follows:

```
register
list name

Contains the address (register) or the name (list name) of the TRPL parameter list.
```
ABEND codes

If the installation has requested to ABEND the program that invokes the tracking service, the task will be ABENDed with an ABEND code of X'077' and a reason code of X'34'. No dump will be taken for this ABEND and the program will continue normally. If a dump is needed, a SLIP trap can be set for this ABEND. An example of the command to set this SLIP trap is:

```
SLIP SET,ENABLE,ID=TRAK,COMP=077,REASON=34,ACTION=SVCD,END
```

To help in identifying the invoker of the tracking service, the following registers will be set. Note that this information is the same as is recorded with the instance and displayed in message CNZ1001I.

- **R2** Contents of TRPL_Track_Data.
- **R3** Address of the CNZTRKR parameter list (mapped by CNZTRPL) that the tracking facility was processing.
- **R4** Address of the jobname that the caller of the tracking facility was running under.
- **R5** Address of the program name that called the service which invoked the tracking facility.
- **R6** Address where the event being tracked occurred.
- **R7** Offset within the program where the event being tracked occurred.
- **R8** ASID of the program that causes the event to be tracked.

Return and reason codes

When the CNZTRKR macro returns control to your program, register 15 contains one of the following hexadecimal return codes. Mapping macro CNZTRPL provides names for each of these Return and Reason codes.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                      | **Meaning**: The recording of an instance completed successfully.  
**Action**: None. |
| 04                      | 04                      | **Meaning**: The request was to record an instance but the maximum number of recorded instances has been reached.  
**Action**: Contact the operator and have him or her recycle the tracking facility by issuing a SETCON TRACKING=OFF operator command followed by a SETCON TRACKING=ON command. |
| 04                      | 08                      | **Meaning**: The request was to record an instance but the tracking facility is not active.  
**Action**: If you need to have the tracking facility active, contact your operator and request that the tracking facility be made active by issuing a SETCON TRACKING=ON operator command. |
| 0C                      | xx                      | **Meaning**: There was an error with the TRPL parameter list.  
**Action**: None. |
## CNZTRKR macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| **0C** 04               |                         | **Meaning:** The acronym in the parameter list was not valid (it must be “TRPL”) or the **version level** was not supported.  

**Action:** Correct the acronym or **version level** in the parameter list and issue CNZTRKR again. |
| **0C** 08               |                         | **Meaning:** The track information provided in the TRPL parameter list was all blank or all hexadecimal zeros.  

**Action:** Make any necessary corrections to your code and issue CNZTRKR again. |
| **0C** 0C               |                         | **Meaning:** There was an error attempting to access the TRPL parameter list. The TRPL address you provided may not have been valid or pointed to storage that the CNZTRKR service could not access.  

**Action:** Make any necessary corrections to your code and issue CNZTRKR again. |
| **10** xx               |                         | **Meaning:** This return code is for IBM diagnostic purposes only.  

**Action:** Record the return and reason codes and supply them to the appropriate IBM support personnel. |
| **10** 04               |                         | **Meaning:** A recovery environment could not be established.  

**Action:** Record the return and reason codes and supply them to the appropriate IBM support personnel. |
| **10** 08               |                         | **Meaning:** A serialization environment could not be established.  

**Action:** If this instance is important to be recorded, you can reissue the request. Serialization may now be able to be obtained. |
| **10** 0C               |                         | **Meaning:** An ABEND occurred in the CNZTRKR service during the processing of your request.  

**Action:** Notify your system programmer. |
Chapter 29. CONVCON — Retrieve console information

Description

IBM suggests using the CnzConv service to retrieve console information. As of z/OS V1R8 and later releases, the CONVCON service will no longer be enhanced. Future enhancement will be provided only on the CnzConv service. For more information about the CnzConv macro, see Chapter 26, “CnzConv -- Convert console name and ID,” on page 129.

Application programmers can retrieve information about MCS, SMCS, or extended MCS consoles by using the CONVCON macro.

You can use CONVCON to:
- Determine the name of a console when you specify the ID
- Determine the ID of a console when you specify the name
- Validate a console name or console ID
- Validate that a console area ID is syntactically correct
- Check if a console is active.

You must initialize a parameter list as input. See z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/ for a map of the CONVCON parameter list, called CONV, which is mapped by IEZVG200. See z/OS MVS Programming: Assembler Services Guide to determine which fields to initialize.

Environment

The requirements for the caller are:

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

Programming requirements

Before issuing CONVCON, do the following:

- Include the IEZVG200 mapping macro in your program.
- Obtain storage for the CONVCON parameter list. CONVGLEN in IEZVG200 contains the length of the parameter list. The parameter list can be in any type of storage.
- Clear the entire parameter list by setting it to zeros. If you reuse it, clear it before each reuse.
- Initialize fields in the parameter list mapped by macro IEZVG200. Depending on the task for which you are invoking CONVCON, you need to initialize a combination of different fields.

You must initialize the following fields no matter what task you perform:

- **CONVACRO** The CONV acronym
CONVCON macro

CONVRSN  The current version level of the parameter list. The parameter list contains valid values in CONVRID.

The following describes the remaining parameter list fields. Depending on the task you choose, these fields are input fields, output fields, or both input and output fields. Use the information in the z/OS MVS Programming: Assembler Services Guide to determine which of these fields are input fields, and which are output fields.

CONVFLGS  A 1-byte flag field that indicates whether you are supplying the console name in CONVFLD (flag CONVPFLD) or the console ID (flag CONVPID) in CONVID. Set only the first bit on to indicate the console name; set only the second bit on to indicate the console ID.

CONVFLD  A 10-byte field containing the console name or console name with the area ID. The installation defines console names at initialization time in the CONSOLxx member of Parmlib. You can use the DISPLAY command to receive a list of defined names. Console area IDs can be only one character, A through K or Z. If you specify a console with an area ID, separate the name and area ID by a hyphen, left-justify it, and pad it to the right with blanks. Examples of valid console names with area IDs are:
- DATA-a
- DATADATA-a

Examples of incorrect names with area IDs and the reasons are:
- DATA-abc - has an area ID with more than one character
- DATA a - has a blank instead of a hyphen between the console name and the area ID

CONVAREA  A one-character input field containing a console area ID. Valid IDs are 'A' - 'K' and 'Z', and are only validated for MCS or SMCS consoles.

CONVRSN  A reason code explaining return codes 0, 4, or 8.

CONVNAME  An 8-byte field containing the console name received as output when you specify the console ID as input. The console name can be up to eight characters. If the name has fewer than eight characters, the name is left-justified and padded to the right with blanks.

CONVID  A 4-byte console ID. If you specified the name of the console in CONVFLD, on return, CONVID contains the ID of the same console. If you specify the ID of the console in CONVID, CONVCON will return the name of the console in CONVNAME. The system assigns console IDs.

CONVGFLG  Set flag CONVNPAR on in this field only if you want CONVCON to omit any area ID processing. If you do set the flag on, CONVCON:
- Ignores an area ID in CONVAREA
- Assumes the entire field is a console name, and issues return code 'X'08' if you included an area ID.

CONVSYSN  If the console name or ID that you specified is active, CONVCON places the name of the system to which the console is attached in CONVSYSN. If the console is not active, this field contains blanks.
CONVCON macro

CONVSMCS  Output flag that indicates that the console specified is an SMCS console.

**Programming considerations**

The CONVID 4-byte console id field can be used to determine if a console is extended MCS or MCS/SMCS. The first byte, mapped by CONVCLAS, is non-zero for an EMCS console. The first byte is zero for MCS and SMCS consoles. The CONVSMCS bit can be used to determine if a console is an SMCS console. This bit will always be on for SMCS consoles, and will be off for other types of consoles.

**Restrictions**

There are six reserved console names. Do not use any of the following as console names:
- HC
- LOGON
- LOGOFF
- OPERLOG
- SYSLOG
- UNKNOWN

**Input register information**

Before issuing CONVCON, 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Address of the CONV parameter list</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 CONVCON macro is written as follows:

```assembly
name

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

b One or more blanks must precede CONVCON.

CONVCON

b One or more blanks must follow CONVCON.

**Parameters**
The parameters are explained as follows:

<table>
<thead>
<tr>
<th>register</th>
<th>register: General purpose register (2) - (12).</th>
</tr>
</thead>
<tbody>
<tr>
<td>list name</td>
<td>list name: RX-type address.</td>
</tr>
<tr>
<td>,RTNCODE=ReturnCode</td>
<td>ReturnCode: RX-type address or register (2)-(12).</td>
</tr>
<tr>
<td>,RSNCODE=ReasonCode</td>
<td>ReasonCode: RX-type address or register (2)-(12).</td>
</tr>
</tbody>
</table>

**ABEND codes**

**077**

**Return and reason codes**
When the CONVCON macro returns control to your program, the following table documents the possible return and reason codes. You can use the RTNCODE= and the RSNCODE= parameter on the CONVCON macro to save the return code and reason code into a variable or register. The return code is available in Register 15 and the reason code is available in Register 0 and CONVRSN when the CONVCON macro returns. Here are the possible return and reason codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                     | **Meaning:** The input console is active and the area ID (if specified) is syntactically valid.  
                          |                        | **Action:** None. |
| 00                      | 0C                     | **Meaning:** Program error. The input console is active and the area ID specified is not syntactically valid.  
                          |                        | **Action:** Correct the area ID specification. The area ID must be a letter between A-K or Z. |
| 00                      | 10                     | **Meaning:** Program error. The input console is active and the area ID was either not specified after the dash or additional non-blank characters were specified after the area ID in CONVFLD.  
                          |                        | **Action:** Correct the area ID specification. The area ID must be a letter between A-K or Z. |
### CONVCON macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04 00                   |                         | **Meaning:** Environmental error. The input console is inactive and the area ID (if specified) is syntactically valid.  
**Action:** Messages cannot be sent to this console. You must direct messages elsewhere. |
| 04 0C                   |                         | **Meaning:** Program error. The input console is inactive and the area ID specified is not syntactically valid.  
**Action:** Messages cannot be sent to this console. You must direct messages elsewhere. Correct the area ID specification. The area ID must be a letter between A-K or Z. |
| 04 10                   |                         | **Meaning:** Program error. The console is not active, but the requested console information was obtained. The specified area ID does not comply with syntax requirements. The area ID must be in the range between A-K, or Z.  
**Action:** Messages cannot be sent to this console. You must direct messages elsewhere. Correct the area ID specification. The area ID must be a letter between A-K or Z. |
| 08 00                   |                         | **Meaning:** Program error. The console name specified is not valid, for one of the following reasons:  
- No console with the specified name exists.  
- You specified an area ID with the console name, but you also set flag CONVNPAR in the CONVGFLG field in the CONV parameter list.  
- You specified a console name with more than 8 characters.  
**Action:** Take the action number corresponding to the meaning number.  
- Change the console name to one that is defined in the sysplex.  
- Remove the area ID after the console name, or turn off the CONVNPAR in the CONV parameter list.  
- Correct the console name. |
| 08 08                   |                         | **Meaning:** Program error. The console name specified contains invalid syntax.  
**Action:** Correct the syntax of the console name and resubmit the request. |
| 08 0C                   |                         | **Meaning:** Program error. You specified a reserved console name.  
**Action:** Correct the problem and resubmit the request. |
<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C                      | N/A                     | **Meaning**: Program error. You specified an incorrect console ID on input.  
**Action**: Specify a valid 4-byte console ID. Correct the problem and resubmit the request. |
| 10                      | N/A                     | **Meaning**: Environmental error. The CONVCON service is not available.  
**Action**: Resubmit the request at a later time. |
| 14                      | N/A                     | **Meaning**: System error. This return code is for IBM diagnostic purposes only.  
**Action**: Record the return code and supply it to the appropriate IBM support personnel. |
| 18                      | N/A                     | **Meaning**: Program error. CONVCON processing completed unsuccessfully. You did not specify whether a console name or a console ID was being supplied as input.  
**Action**: Ensure that exactly one of the console input flags in field CONVFLGS is set and resubmit the request. |
| 1C                      | N/A                     | **Meaning**: Program error. CONVCON processing completed unsuccessfully. You specified both the console name and console ID values in CONVFLGS.  
**Action**: Ensure that only one of the console input flags in field CONVFLGS is set and resubmit the request. |
| 20                      | N/A                     | **Meaning**: Program error. CONVCON processing completed unsuccessfully. The CONV acronym was missing in the CONV parameter list.  
**Action**: Ensure that you are correctly referencing the parameter list when issuing CONVCON, and that the parameter list is correct. Resubmit the request. |
| 24                      | N/A                     | **Meaning**: Program error. CONVCON was called while holding a lock.  
**Action**: Correct the program to invoke CONVCON while no locks are held. |
| 28                      | N/A                     | **Meaning**: The CONVCON service was invoked in an incorrect environment.  
**Action**: Invoke the CONVCON service in the correct environment. |

**Example**
A typical application of CONVCON would be in an MPF message exit. A message exit could be built to reroute a message to an active console in your system or sysplex. The example below has been coded with the following in mind:

- If CONVCON indicated the console is active and you want to send the message only to that active console, add the necessary code at the CONTINUE label to send the message only to the active console.
CONVCON macro

- If CONVCON indicated the console was inactive and you want to route this message to a different console, add the necessary code to the NOTACTIVE label.

And this example assumes:
- That you would have function specified at the labels referenced as locations of branch instructions (block comments are also in the example showing where these would be).
- That you are not reusing your CONV parameter list. If you need to issue subsequent CONVCON request in other areas of the code, you must clear the CONV parameter list and initialize it to perform the subsequent query.

CALLCONV CSECT
ZERO EQU 0
REG2 EQU 2
REG4 EQU 4
REG12 EQU 12
REG13 EQU 13
REG14 EQU 14
REG15 EQU 15
RCINACT EQU 4
* THIS EXAMPLE CALLS CONVCON TO DETERMINE THE STATUS OF
* A CONSOLE
STM REG14,REG12,12(REG13) SAVE REGISTERS OF CALLER
BASR REG12,0 ESTABLISH BASE REGISTER
USING *,REG12 GET MODULE ADDRESSABILITY
LA REG2,CONVGLEN AMOUNT OF STORAGE TO GET
STORAGE OBTAIN,LENGTH=(REG2),ADDR=(REG4)
USING CONV,REG4 GET ADDRESSABILITY TO CONV
XC 0(CONVGLEN,REG4),0(REG4) CLEAR PARAMETER LIST
MVC CONVACRO,ACNMCONV PUT ACRONYM INTO PARAMETER LIST
MVI CONVRVRSN,CONVRID PUT VERSION INTO PARAMETER LIST
OI CONVFLD,CONSVNAME TURN ON CONSOLE NAME FLAG
MVC CONVFLD,CONSNAME PUT CONSOLE NAME IN PARAMETER X LIST
CONVCON (REG4) CALL CONVCON
LTR REG15,REG15 IS THE CONSOLE ACTIVE?
BZ CONTINUE YES, GOTO CONTINUE
CHI REG15,RCINACT IS THE CONSOLE INACTIVE?
BE NOTACTIVE YES, GOTO NOTACTIVE
B EXIT END PROCESSING

CONTINUE EQU *
***********************************************************************
* HERE YOU WOULD PROVIDE SUPPORT FOR THE ACTIONS YOU WANTED *
* TO TAKE IF THE CONSOLE WAS ACTIVE. *
***********************************************************************

NOTACTIVE EQU *
***********************************************************************
* HERE YOU WOULD PROVIDE SUPPORT FOR THE ACTIONS YOU WANTED *
* TO TAKE IF THE CONSOLE WAS INACTIVE. *
***********************************************************************

EXIT EQU *
STORAGE RELEASE,LENGTH=CONVGLEN,ADDR=(REG4)
DROP REG4 DROP ADDRESSABILITY TO CONV
LM REG14,REG12,12(REG13) RESTORE REGISTERS OF CALLER
BR REG14 RETURN TO CALLER

ACNMCONV DC C'CONV' CONVCON ACRONYM
CONSNAME DC C'CONSOLE1' CONSOLE NAME WITH AN AREA
IEZVG200 CONVCON PARAMETER LIST
END
CONVCON macro
Chapter 30. CONVTOD — Convert to time-of-day clock format

Description

The CONVTOD macro accepts a time and date value in several different formats and converts it to time-of-day (TOD) clock format. The clock format can be either the basic time-of-day (TOD) or the extended time-of-day (ETOD).

- **TOD** — Unsigned 64-bit binary number
- **ETOD** — Unsigned 128-bit binary number

See [z/OS MVS Programming: Assembler Services Guide](https://www.ibm.com) and [z/Architecture Principles of Operation](https://www.ibm.com) for information comparing the formats of the TOD and ETOD.

The input time and date formats are compatible with those returned by the STCKCONV and TIME macros.

Environment

The requirements for the caller are:

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

Programming requirements

If the program is in AR mode, issue the SYSSTATE ASCENV=AR macro before CONVTOD. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

None.

Input register information

Before issuing the CONVTOD 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
CONVTOD Macro

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 CONVTOD macro is written as follows:

```
name name
\/\slash
CONVTOD
\/\slash
CONVVAL=convval
,TODVAL=todval
,ETODVAL=etodval
,TIMETYPE=DEC
,TIMETYPE=BIN
,TIMETYPE=MIC
,DATE\_TYPE=YYDDD
,DATE\_TYPE=YYYYDDD
,DATE\_TYPE=DDMYYDDD
,DATE\_TYPE=DDMMYYYY
,DATE\_TYPE=MMDDYYYY
,DATE\_TYPE=YYYYMMDD
,OFFSET=offset value
```

Parameters

The parameters are explained as follows:

- **name**: Symbol. Begin name in column 1.
- One or more blanks must precede CONVTOD.
- One or more blanks must follow CONVTOD.
- **CONVVAL=convval**: `convval`: RX-type address or register (2) - (12).
- **TODVAL=todval**: `todval`: RX-type address or register (2) - (12).
- **ETODVAL=etodval**: `etodval`: RX-type address or register (2) - (12).
- **TIMETYPE=DEC**: Default: TIMETYPE=DEC.
- **TIMETYPE=BIN**:
- **TIMETYPE=MIC**:
- **DATE\_TYPE=YYDDD**: Default: DATE\_TYPE=YYDDD.
- **DATE\_TYPE=YYYYDDD**:
- **DATE\_TYPE=DDMYYDDD**:
- **DATE\_TYPE=DDMMYYYY**:
- **DATE\_TYPE=MMDDYYYY**:
- **DATE\_TYPE=YYYYMMDD**:
- **OFFSET=offset value**: `offset value`: RX-type address or register (2) - (12).
  Default: OFFSET=X'0000000F'.

CONVVAL = *convval*

Specifies a 16-byte storage area in which you will enter the time and date values to be converted. The storage area must begin on a word boundary. The first two words contain the time of day and the third word contains the date in the formats specified by the TIMETYPE and DATETYPE parameters. Set the fourth word to 0 before issuing CONVTOD.

The earliest valid date is January 1, 1900 and the latest valid date is September 17, 2042.

**TODVAL = *todval***

Specifies an 8-byte storage area where the TOD-clock-formatted value is to be returned. The storage area must begin on a word boundary.

**ETODVAL = *etodval***

Specifies a 16-byte storage area where the ETOD-clock-formatted value is to be returned. The storage area must begin on a word boundary.

Only one of either TODVAL or ETODVAL can be specified.

**TIMETYPE = DEC**

**TIMETYPE = BIN**

**TIMETYPE = MIC**

Specifies the format of the input time value:

- **DEC**
  - Unsigned packed decimal digits representing a time value in the form
    - HH is hours, based on a 24-hour clock
    - MM is minutes
    - SS is seconds
    - t is tenths of a second
    - h is hundredths of a second
    - m is milliseconds
    - i is ten-thousandths of a second
    - j is hundred-thousandths of a second
    - u is microseconds.

  **Note:** HHMMSSth must be in the first word with the remainder left-justified in the second word. Set the unused part of the second word to zeros.

- **BIN**
  - Unsigned 32-bit binary number representing a time value as an unsigned binary number in which the low-order bit represents 0.01 of a second. Obtain but do not use the second word.

- **MIC**
  - Unsigned 64-bit binary number representing a time value in microseconds. Bit 51 represents 1 microsecond.

**DATETYPE = YYDDD**

**DATETYPE = YYYYDDD**

**DATETYPE = DDMMYYYY**

**DATETYPE = MMDDYYYY**

**DATETYPE = YYYYMMDD**

Specifies the format of the input date value:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Format of input date</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYDDD</td>
<td>0CYYDDDFF</td>
</tr>
<tr>
<td>YYYYDDD</td>
<td>0YYYYYDDD</td>
</tr>
<tr>
<td>DDMMYYYY</td>
<td>DDMMYYYY</td>
</tr>
<tr>
<td>MMDDYYYY</td>
<td>MMDDYYYY</td>
</tr>
<tr>
<td>YYYYMMDD</td>
<td>YYYYMMDD</td>
</tr>
</tbody>
</table>
CONVTOD Macro

Where:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OC</td>
<td>is the century - 00 represents 19YY, 01 represents 20YY</td>
</tr>
<tr>
<td>F</td>
<td>is a sign to enable the date to be unpacked</td>
</tr>
<tr>
<td>YY</td>
<td>is the last two digits of the year</td>
</tr>
<tr>
<td>YYYY</td>
<td>is the year</td>
</tr>
<tr>
<td>DDD</td>
<td>is the day of the year (Julian date)</td>
</tr>
<tr>
<td>DD</td>
<td>is the day of the month</td>
</tr>
<tr>
<td>MM</td>
<td>is the month of the year</td>
</tr>
</tbody>
</table>

OFFSET=offset value

Specifies a 4-byte storage area containing a packed decimal number of the form 000HHMMX, where X is the sign (D for a negative number, F for a positive number). The offset value is added to the input time. The offset value is generally the difference between Greenwich Mean Time and local time but it can be any desired value. The default value is X'0000000F'.

ABEND codes

None.

Return and reason codes

The following table describes CONVTOD’s return codes, their meanings, and any recommended actions you should take. Return codes are listed in hexadecimal with their decimal value shown in parentheses.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (00)</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
<tr>
<td>0C (12)</td>
<td>Meaning: Unsuccessful completion. CONVTOD encountered an unexpected error. Action: Record the return code and supply it to the appropriate IBM support personnel.</td>
</tr>
<tr>
<td>10 (16)</td>
<td>Meaning: Unsuccessful completion. The caller’s parameter list was not addressable. Action: Verify that the pointer to the parameter list contains a valid address and that CONVTOD is being invoked in a valid addressing mode.</td>
</tr>
<tr>
<td>14 (20)</td>
<td>Meaning: Unsuccessful completion. The time, date, or offset parameter value was not valid. Action: Verify that the input parameters have been initialized correctly.</td>
</tr>
</tbody>
</table>

Example 1

Convert a time expressed as microseconds and a date expressed as month-day-year to TOD clock format using the specified offset value:

```assembler
CONVTOD CONVVAL=INAREA,TODVAL=OUTAREA,TIMETYPE=MIC, * DATETYPE=MMDDYYYY,OFFSET=PLUS1
INAREA DS 0F
DC X'00009047F3070000' INPUT TIME IN MIC FORMAT
DC X'05171990' INPUT DATE IN MMDDYYYY FORMAT
DS 'F'0' UNUSED FOURTH WORD
PLUS1 DC X'0000100F' +1 HOUR OFFSET VALUE
OUTAREA DS 2F AREA FOR OUTPUT TOD CLOCK VALUE
```
Example 2

Convert a time expressed as a decimal value and a date expressed as the Julian date to TOD clock format using the specified offset value:

```
CONVTOD CONVVAL=INAREA,TODVAL=OUTAREA,TIMETYPE=DEC, *
   DATETYPE=YYDDD,OFFSET=MINUSFIV
INAREA DS 0F
   DC X'1045301535120000' INPUT TIME IN DEC FORMAT
   DC X'0090137F' INPUT DATE IN YYDDD FORMAT
   DS F'0' UNUSED FOURTH WORD
MINUSFIV DC X'00000500D' -5 HOUR OFFSET VALUE
OUTAREA DS 2F AREA FOR OUTPUT TOD CLOCK VALUE
```

Example 3

Convert a time expressed as a binary value and a date expressed as year-month-day to TOD clock format using the default offset value:

```
LA 3,INAREA STORE INPUT AREA ADDRESS
LA 11,OUTAREA STORE OUTPUT AREA ADDRESS
LA 6,PLIST STORE PARAMETER LIST ADDRESS
CONVTOD CONVVAL=(3),TODVAL=(11),TIMETYPE=BIN,DATETYPE=YYYYMMDD*,
   ,MF=(E,(6))

PLIST CONVTOD MF=L GENERATE PARAMETER LIST STORAGE
INAREA DS 0F
   DC X'003B18F700000000' INPUT TIME IN BIN FORMAT
   DC X'19900517' INPUT DATE IN YYYYMMDD FORMAT
   DS F'0' UNUSED FOURTH WORD
OUTAREA DS 2F AREA FOR OUTPUT TOD CLOCK VALUE
```

CONVTOD—List form

Use the list form of the CONVTOD macro together with the execute form of the macro for programs that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to store the parameters.

Syntax

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

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

b One or more blanks must precede CONVTOD.

CONVTOD

b One or more blanks must follow CONVTOD.

MF=L
```
CONVTOD Macro

Parameters

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

MF=L

Specifies the list form of the CONVTOD macro. Do not specify any other keywords with MF=L. Precede the macro invocation with a name in column 1 to label the generated parameter list so you can refer to it.

CONVTOD—Execute form

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

Syntax

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

```
name name  name: Symbol. Begin name in column 1.
\ 
CONVTOD
\ 
CONVVAL=convval  convval: RX-type address or register (2) - (12).
\ ,TODVAL=todval  todval: RX-type address or register (2) - (12).
\ ,ETODVAL=etodval  etodval: RX-type address or register (2) - (12).
\ ,TIMETYPE=DEC  Default: TIMETYPE=DEC.
\ ,TIMETYPE=BIN
\ ,TIMETYPE=MIC
\ ,DATETYPE=YYDDD  Default: DATETYPE=YYYYDDD.
\ ,DATETYPE=YYYYDDD
\ ,DATETYPE=DDMMYYYY
\ ,DATETYPE=MMDDYYYY
\ ,DATETYPE=YYYYMMDD
\ ,OFFSET=offset value  offset value: RX-type address or register (2) - (12).
\ Default: OFFSET=X'0000000F'.
\ ,MF=(E,list addr)  list addr: RX-type address or register (1) - (12).
```
Parameters

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

\[ \text{MF=}(E, \text{list addr}) \]

Specifies the execute form of the CONVTOD macro. \text{list addr} specifies the area that the system uses to store the parameters.
CONVTOD Macro
Chapter 31. CPOOL — Perform cell pool services

Description

The CPOOL macro performs the following functions:
- Creates a cell pool, where each cell is of the size you specify
- Obtains or returns a cell to the cell pool
- Deletes the previously built cell pool
- Places the starting and ending addresses of the cell pool extents in a buffer.

Problem-state programs running under PSW key 8-15 can obtain cell pools from subpools 0-127, 131, and 132. Before obtaining storage, be sure to read the information on subpools in "Virtual Storage Management" in z/OS MVS Programming: Assembler Services Guide.

Environment

The requirements for the caller are:

Minimum authorization:
- For subpools 0-127: problem state and PSW key 8-15
- For subpools 131 and 132: APF authorization or a PSW key mask (PKM) that allows the caller to switch into the storage key of the storage to be obtained.

Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN.
AMODE: 24- or 31-bit
ASC mode: For LIST requests, primary or secondary. For all other requests, primary.
Interrupt status: Enabled or disabled for I/O and external interrupts.
Locks: The following locks must be held or must be obtainable by CPOOL:
- If the caller is not running in cross-memory mode, the LOCAL lock of the currently addressable address space.
- If the caller is running in cross-memory mode, the CML lock of the currently addressable address space.
Control parameters: Must reside in the primary address space and may reside in storage above 16 megabytes if the caller is in 31-bit addressing mode.

Programming requirements

None.

Restrictions

None.

Input register information

The CPOOL macro is sensitive to the SYSSTATE macro with the OSREL=ZOSV1R6 parameter
- If the caller has issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter (Version 1 Release 6 of z/OS or later) before issuing the CPOOL macro with the BUILD, DELETE, LIST, or REGS=SAVE parameters, the caller...
CPOOL macro

does not have to place any information into any general purpose register (GPR) unless using it in register notation for a particular parameter, or using it as a base register.

- If the caller has not issued the SYSSTATE macro with the OSREL=ZOSV1R6 parameter before issuing the CPOOL macro with the BUILD, DELETE, LIST, or REGS=SAVE parameters, the caller must ensure that the following general purpose register (GPR) contains the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of an 72-byte save area</td>
</tr>
</tbody>
</table>

Before issuing the CPOOL macro with the GET, FREE, or REGS=USE parameters, the caller is not required to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller from CPOOL BUILD, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Contains the cell pool id.</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller from CPOOL GET, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as work registers for the system.</td>
</tr>
<tr>
<td>1</td>
<td>For an UNCOND request or a successful COND request, contains the address of the obtained cell. For an unsuccessful COND request, contains a zero.</td>
</tr>
<tr>
<td>2-4</td>
<td>If REGS=SAVE is specified, the registers remain unchanged. Otherwise, used as work registers by the system.</td>
</tr>
<tr>
<td>5-13</td>
<td>If LINKAGE=SYSTEM, REGS=SAVE, COND REGS=USE, or MULTIHDR=YES is specified, the registers remain unchanged. Otherwise, the registers are used as work registers by the system.</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller from CPOOL FREE, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-3</td>
<td>If REGS=SAVE is specified, the registers remain unchanged. Otherwise, used as work registers by the system.</td>
</tr>
<tr>
<td>4-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller from CPOOL DELETE, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller from CPOOL LIST, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>
CPOOL macro

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

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system.</td>
</tr>
</tbody>
</table>

Performance implications

The CPOOL macro offers better performance than GETMAIN-FREEMAIN and STORAGE for obtaining and releasing many identically sized storage areas.

Syntax

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

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

b
One or more blanks must precede CPOOL.

CPOOL

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

Valid parameters (Required parameters are underlined)

- **BUILD**
  - PCELLCT,SCELLCT,CSIZE,SP,BNDRY,LOC,CPID,HDR
- **GET**
  - UNCOND,COND,CPID,CELL,REGS
- **FREE**
  - CPID,CELL,REGS
- **DELETE**
  - CPID
- **LIST**
  - CPID,WORKAREA

- **,UNCOND**
  - Default: UNCOND
- **,U**
- **,COND**
- **,C**

- **,PCELLCT=primary cell count**
  - cell count: Symbol, decimal number, or register (0), (2) - (12).

- **,SCELLCT=secondary cell count**
  - Default: PCELLCT

- **,CSIZE=cell size**
  - cell size: Symbol, decimal number, or register (0), (2) - (12).

- **,SP=subpool number**
  - subpool number: Symbol, decimal number, or register (0), (2) - (12).
  - Default: SP=0
CPOOL macro

, BNDRY=DWORD

| Default: BNDRY=DWORD
| The default value depends on the specified CSIZE value. If CSIZE is a
| multiple of 8, cells reside on double boundaries (BNDRY=DWORD). If CSIZE
| is multiple of 4, cells reside on word boundaries. If CSIZE is not a multiple of
| 4 or 8, cells do not reside on a particular boundary.

, BNDRY=QWORD

, LOC=24
| Default: LOC=RES
| LOC=31
| LOC=(31,31)
| LOC=(31,64)
| LOC=(31,PAGEFRAMESIZE1MB)
| LOC=RES
| LOC=(RES,31)
| LOC=(RES,64)

, CPID=pool id
| pool id: RX-type address or register (0), (2) - (12).

, CELL=cell addr
| cell addr: RX-type address or register (2) - (12).

, KEY=key number
| key number: Decimal numbers 0-15 or register (0), (2) - (12).
| Default: The default depends on which subpool you specify. See the
| discussion of subpool handling in [z/OS MVS Programming: Assembler][Services Guide] for information on storage keys for specific subpools.

, HDR=hdr
| hdr: Character string enclosed in single quotation marks, RX-type address, or
| register (0), (2) - (12).
| Default: ‘CPOOL CELL POOL’

, REGS=SAVE
| Default: REGS=SAVE

, REGS=USE

, WORKAREA=(workarea,length)
| workarea: Symbol, RX-type address, or register (0), (2) - (12).
| length: Symbol or decimal number.

Parameters

The parameters are explained as follows:

BUILD
GET
FREE
DELETE
LIST

Specifies the cell pool service to be performed.

BUILD creates a cell pool in a specified subpool by allocating storage and
chaining the cells together. It returns an identifier (CPID) to be used with GET,
FREE, and DELETE requests. Therefore, specify BUILD before you specify
GET, FREE, or DELETE.

GET attempts to obtain a cell from the previously built cell pool. This request
can be conditional or unconditional as described under the UNCOND/COND
keyword.
CPOOL macro

FREE returns a cell to the cell pool. Do not try to free a cell that has not been obtained (through the GET service) or free a cell for a second time.

DELETE deletes a previously built cell pool and frees storage for the initial extent, all secondary extents, and all pool control blocks.

LIST places the beginning and ending addresses of the extents of a cell pool in a work area provided by the caller.

,UNCOND
,U
,COND
,C
When used with GET specifies whether the request for a cell is conditional or unconditional.

If you specify COND or C and no more free cells are available in the cell pool, the system returns to the caller without a cell. The system places a zero in the field specified by the CELL parameter.

If you specify UNCOND or U and no more free cells are available in the cell pool, the system obtains more storage for the cell pool. CPOOL then obtains a new cell for the caller. An unconditional CPOOL GET request fails only if enough storage is not available to extend the cell pool.

,PCELLCT=primary cell count
Specifies the number of cells expected to be needed in the initial extent of the cell pool.

,SCELLCT=secondary cell count
Specifies the number of cells expected to be in each secondary or noninitial extent of the cell pool.

,CSIZE=cell size
Specifies the number of bytes in each cell of the cell pool. If CSIZE is a multiple of 8, the cell resides on doubleword boundaries. If CSIZE is a multiple of 4, the cell resides on word boundaries. The minimum value of CSIZE is 4 bytes.

When the specified cell size is less than 256 bytes, the number of elements allocated to an extent may be more than what is expected and may hold more elements than would have fit in an extent of the specified size. This occurs because each extent is allocated to have a length that is a multiple of 256 bytes.

,SP=subpool number
Specifies the subpool from which the cell pool is to be obtained. If a register or variable is specified, the subpool number is taken from bits 24-31. The valid subpool numbers are 0-127, 131, and 132.

,BNDRY=DWORD
,BNDRY=DWORD
Specifies whether each cell must be on at least a doubleword boundary (DWORD) or a quadword (16-byte) boundary (QWORD). The default depends on the value that is specified for CSIZE.

Notes:
1. When BNDRY=DWORD is explicitly specified, a CSIZE value that is multiple of 8 must also be specified to ensure that each cell is on at least a doubleword boundary.
2. When BNDRY=QWORD is explicitly specified, a CSIZE value that is multiple of 16 must also be specified to ensure that each cell is on at least a quadword boundary.
CPOOL macro

LOC=24
LOC=31
LOC=(31,31)
LOC=(31,64)
LOC=(31,PAGEFRAMESIZE1MB)
LOC=RES
LOC=(RES,31)
LOC=(RES,64)

Specifies the location of virtual storage and central storage for the cell pool. The location of central storage using this parameter is guaranteed only after the storage is fixed.

LOC=24 indicates that central and virtual storage are to be located below 16 megabytes. LOC=24 must not be used to allocate disabled reference (DREF) storage.

**Note:** Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW is still supported, but IBM recommends using LOC=24 instead.

LOC=31 and LOC=(31,31) indicate that virtual and central storage can be located anywhere below 2 gigabytes.

**Note:** Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC =31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64) indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

LOC=(31,PAGEFRAMESIZE1MB) indicates that virtual storage is to be located below 2 gigabytes and central storage can be backed anywhere in 64-bit storage, preferably by 1 megabyte page frames. When specifying LOC=(31,PAGEFRAMESIZE1MB) during CPOOL BUILD:

- The only xx sub-parameter value that can be validly specified in combination with the PAGEFRAMESIZE1MB yy sub-parameter on the LOC statement of the CPOOL BUILD macro is 31:

  LOC=(31,PAGEFRAMESIZE1MB)

- PAGEFRAMESIZE1MB indicates that the preferred page frame size for the CPOOL virtual storage range is 1MB.

  **Attention:** PAGEFRAMESIZE1MB is a page size preference only; it does not guarantee that the virtual storage range will be backed by large pages.

- There are no requirements that the 31-bit virtual storage obtained be large page aligned or that it be a multiple of the specified large page size.

- The LOC(31,PAGEFRAMESIZE1MB) parameter has no effect on other parameters that can be specified on CPOOL BUILD requests.

LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be allocated below 16 megabytes; if the issuer resides above 16 megabytes, virtual and central storage can be located anywhere.

LOC=(RES,31) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere below 2 gigabytes. In either case, central storage can be located anywhere below 2 gigabytes.
CPOOL macro

**Note:** Specifying LOC=(RES,ANY) is the same as specifying LOC=(RES,31). LOC=(RES,ANY) is still supported, but IBM recommends using LOC=(RES,31) instead.

LOC=(RES,64) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere in 64-bit storage. In either case, central storage can be located anywhere in 64-bit storage.

**Note:** Callers executing in 24-bit addressing mode could perform BUILD request services for cell pools located in storage above 16 megabytes but below 2 gigabytes by specifying LOC=31 or LOC=(31,31).

**,CPID=pool id**

Specifies the address or register containing the cell pool identifier that is returned to the caller after the pool is created using CPOOL BUILD. The issuer must specify CPID on all subsequent GET, FREE, DELETE, or LIST requests.

**,CELL=cell addr**

Specifies the address or register where the cell address is returned to the caller on a GET or FREE request.

**,KEY=key number**

Specifies the storage key in which storage is to be obtained. The valid storage keys are 0-15. If a register is specified, the storage key is taken from bits 28-31. This parameter is valid only for subpools 131 and 132.

**,HDR=hdr**

Specifies a 24-byte header, which is placed in the header of each initial and secondary extent. The header can contain user-supplied information that would be useful in a dump.

**,REGS=SAVE**, **REGS=USE**

Indicates whether or not registers 2-12 are to be saved for a GET or FREE request. If REGS=SAVE is specified, the registers are saved in a 72-byte user-supplied save area pointed to by register 13. If REGS=USE is specified, the registers are not saved.

**,WORKAREA=(workarea,length)**

Specifies the address of a pointer to the work area (not the address of the work area) and also specifies the length of that area. The length must be at least 1024 bytes. The system places the beginning and ending addresses of the extents of a cell pool in this work area. WORKAREA applies only to the LIST request and is required.

CPOOL LIST might not be able to return all of the beginning address/ending address pairs at once, depending on how many address pairs there are and how large the work area is. Thus, in order to complete a CPOOL LIST request, your program may have to issue CPOOL LIST more than once. If CPOOL LIST uses up all the space in the work area, but still has more information to return, it indicates (with a return code) that there are more address pairs. Your program can then reissue CPOOL LIST to get more information, and keep reissuing CPOOL LIST until all of the information is returned.

CPOOL LIST must be able to tell the difference between the beginning of a request (that is, the first time your program issues CPOOL LIST to get some information about a cell pool) and the continuation of a request (that is, when...
your program issues CPOOL LIST to get more information). Your program tells CPOOL LIST that it is beginning a new request by setting the first bit of word 0 in the work area to 1.

Until your program has obtained all the information about a cell pool that it needs from CPOOL LIST, it should not change the setting of that bit, nor should it issue a GET, FREE, or DELETE request for that cell pool. (If your program does issue a GET or FREE request before it has obtained all of the information it needs from CPOOL LIST, it must begin a new CPOOL LIST request; that is, set the first bit of word 0 to 1 and start all over again. If your program deletes the cell pool, it can no longer issue the CPOOL LIST for that cell pool.)

CPOOL LIST uses the second through fourth words, i.e., words 1—3, in the work area to return information to your program:

- Word 1 contains the return code. See “Return and reason codes” for more information.
- Word 2 contains a pointer to the first starting address/ending address pair in the list of address pairs.
- Word 3 contains the number of address pairs in the list.

**ABEND codes**

The CPOOL macro issues abend code X'C78'. For detailed abend code information, see [z/OS MVS System Codes](https://www.ibm.com/support/knowledgecenter/SSEPGG_1.1.0/com.ibm.doc.mvs_ses81/sect16/mvswedge.htm).

**Return and reason codes**

CPOOL BUILD, DELETE, FREE, and GET,UNCOND have no return codes. If any of these requests fail, CPOOL issues an abend.

For CPOOL GET,COND, the cell address is returned as zero when there are no more cells in the pool.

CPOOL LIST returns a return code in word 1 (bytes 4 through 7) of the work area used to return information to the calling program.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0           | **Meaning**: Successful completion.  
**Action**: None. |
| 1           | **Meaning**: The work area holds all the information that fit but more information remains to be returned.  
**Action**: Reissue the CPOOL LIST request to receive more information. Do not set the first bit of word 0 in the work area to 1 before reissuing the CPOOL LIST request. |
| 2           | **Meaning**: Program error. At least one parameter passed in the CPOOL LIST request was not valid.  
**Action**: Verify that you have coded the CPOOL LIST parameters correctly. Ensure that the work area is at least 1024 bytes. |
Table 5. Hexadecimal Return Codes for CPOOL LIST (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 3           | **Meaning:** Program or system error. The system found a cell pool control block that was either inaccessible or not valid. The work area contains the information CPOOL LIST gathered before encountering the problem.  
**Action:** Verify that the affected cell pool has not been deleted. If the cell pool still exists, inform the system programmer so that a dump can be taken to get more information to supply to IBM support personnel. |

**Example 1**
Create a cell pool containing 40-byte cells from subpool 2. Allow for 10 cells in the initial extent and 20 cells in all subsequent extents of the cell pool.

CPOOL BUILD,PCELLCT=10,SCELLCT=20,CSIZE=40,SP=2

**Example 2**
Unconditionally obtain a cell pool, specifying the pool ID in register 2. Do not save the registers.

CPOOL GET,U,CPID=(2),REGS=USE

**Example 3**
Free a cell specifying the pool ID in register 2 and the cell address in register 3.

CPOOL FREE,CPID=(2),CELL=(3)

**Example 4**
Delete a cell pool, specifying the pool ID in register 2.

CPOOL DELETE,CPID=(2)

**Example 5**
Request that the system place the starting and ending addresses of a cell pool in a buffer. Assume that the cell pool ID has been saved in POOLID.

```
LA 1,WKAREA Get the address of the work area
ST 1,WKPTR And save it (to pass to CPOOL LIST)

* (Note that the first parameter passed with WORKAREA
  * is a pointer to the work area, not the work area itself.)
* OI FLAGBYTE,X'80' Turn on the "first call" flag
LOOP
  LA 13,SAVEAREA Get address of save area in reg 13
  CPOOL LIST,WORKAREA=(WKPTR,1050),CPID=POOLID
  LA 15,2 Get a return code value
  C 15,RCODE Check the return code
  BE USRERROR Branch if there was a user error
* If the return code does not indicate a user error,
  * some information was returned in the work area. Note
  * that if CPOOL LIST found that the first extent it looked
  * at was invalid, the buffer may not actually contain any
  * address pairs (i.e. ENTRIES may contain 0).
* BAL 14,PROCESS Process the information returned
  * by CPOOL LIST
  LA 15,1 Get a return code value
  C 15,RCODE If CPOOL LIST could not return all
  * the information at once,
  BE LOOP Call it again to get more information
```
CPOOL macro

* Data declarations

WKAREA DS 0CL1050 Work area/buffer for CPOOL LIST
FLAGBYTE DS CL1 Byte containing first call flag
DS CL3
RCODE DS F CPOOL LIST return code
BUFPTR DS F Pointer to output buffer
ENTRIES DS F Number of address pairs in buffer
DS CL1034 Control information and address pairs
WKPTR DS F Pointer to the work area
POOLID DS F Cell pool ID
SAVEAREA DS CL72 Register save area for CPOOL LIST

CPOOL-List form

The list form of the CPOOL macro builds a nonexecutable parameter list that can be referred to by the execute form of the CPOOL macro.

Syntax

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

```
name
b
CPOOL
b
```

BUILD

```
,PCELLCT=primary cell count
,SCELLCT=secondary cell count
,CSIZE=cell size
,SP=subpool number
,BNDRY=DWORD
```

cell count: Symbol or decimal number.

Note: PCELLCT must be specified on either the list or the execute form of the macro.

Default: PCELLCT

Default: PCELLCT

Default: CSIZE

Default: CSIZE

Default: CSIZE

Default: CSIZE

The default value depends on the specified CSIZE value. If CSIZE is a multiple of 8, cells reside on double boundaries (BNDRY=DWORD). If CSIZE is a multiple of 4, cells reside on word boundaries. If CSIZE is not a multiple of 4 or 8, cells do not reside on a particular boundary.
CPOOL macro

Parameters

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

,MF=L

Specifies the list form of the CPOOL macro.

CPOOL-Execute form

Syntax

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

```plaintext
name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede CPOOL.

CPOOL

One or more blanks must follow CPOOL.
```

Note: PCELLCT must be specified on either the list or the execute form of the macro.

Default: PCELLCT

```
,PCELLCT=primary cell count

cell count: Symbol, decimal number, or register (0), (2) - (12).

,SCELLCT=secondary cell

Default: PCELLCT
```
CPOOL macro

count

,CSIZE=cell size

   cell size: Symbol, decimal number, or register (0), (2) - (12).

   Note: CSIZE must be specified on either the list or the execute form of the
   macro.

,SP=subpool number

   subpool number: Symbol, decimal number, or register (0), (2) - (12).

   Default: SP=0

,BNDRY=DWORD

   Default: BNDRY=DWORD

   The default value depends on the specified CSIZE value. If CSIZE is a
   multiple of 8, cells reside on double boundaries (BNDRY=DWORD). If CSIZE
   is multiple of 4, cells reside on word boundaries. If CSIZE is not a multiple of
   4 or 8, cells do not reside on a particular boundary.

,BNDRY=QWORD

,L=LOC=24

,L=LOC=31

,L=LOC=(31,31)

,L=LOC=(31,64)

,L=LOC=(31,PAGEFRAMESIZE1MB)

   Default: LOC=RES

   LOC=RES

   LOC=(RES,31)

   LOC=(RES,64)

   LOC=RES

   LOC=(RES,31)

   LOC=(RES,64)

,C_PID=pool id

   pool id: RX-type address or register (0), (2) - (12).

,KEY=key number

   key number: Decimal numbers 0-15 or register (0), (2) - (12).

   Default: The default depends on which subpool you specify. See the
   discussion of subpool handling in [z/OS MVS Programming: Assembler]
   Services Guide for information on storage keys for specific subpools.

,HDR=hdr

   hdr: Character string enclosed in single quotation marks, RX-type address, or
   register (0), (2) - (12).

,MF=(E,ctrl prog)

   ctrl prog: RX-type address or register (0) - (12).


Parameters

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

,MF=(E,ctrl prog)

   Specifies the execute form of the CPOOL macro.
Chapter 32. CPUTIMER — Provide current CPU timer value

Description

The CPUTIMER macro provides the current CPU timer value for this processor. This value consists of the time remaining in a time interval established by the STIMER macro. If there is no outstanding time interval, the value returned by the macro is meaningless.

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 or disabled for I/O and external interrupts
Locks: No locks held
Control parameters: Must be in the primary address space.

Programming requirements

None.

Restrictions

None.

Input register information

The CPUTIMER macro is sensitive to the SYSSTATE macro with the OSREL parameter

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of an 18-word save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
CPUTIMER macro

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 CPUTIMER macro is written as follows:

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

b
  One or more blanks must precede CPUTIMER.

CPUTIMER

b
  One or more blanks must follow CPUTIMER.

TU,stor addr
  Default: TU

MIC,stor addr
  stor addr: RX-type address, or register (1), (2) - (12).

,ERRET=err rtn addr
  err rtn addr: RX-type address, or register (2) - (12).
```

Parameters

The parameters are explained as follows:

```
TU,stor addr
MIC,stor addr
  Specifies the form in which the remaining time interval is to be returned to the caller. This value is returned as an unsigned 64-bit binary number at the address specified by stor addr. stor addr must be the start of a doubleword area on a doubleword boundary and it must be a 31-bit address.

If you specify TU, the timer value is returned to the caller in timer units. The low-order bit of the timer value is approximately equal to 26.04166 microseconds (one timer unit).

If you specify MIC, the timer value is returned to the caller in microseconds. Bit 51 of the timer value is equivalent to 1 microsecond.

The resolution of CPU timer is model dependent. See Principles of Operation for a description of the CPU timer.

,ERRET=err rtn addr
  Specifies the 31-bit address of the routine to be given control when the CPUTIMER function cannot be performed. If you omit this parameter, the CPUTIMER function returns a code in general register 15 indicating why the function failed.
```
CPUTIMER macro

function could not be performed. The error routine executes in the addressing mode of the issuer of the CPUPUTIMER macro and returns control to the caller's address space it saves in register 14.

ABEND codes

None.

Return codes

When the system returns control to your program, GPR 15 contains a return code.

Table 6. Return and Reason Codes for the CPUDUMPER Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: The function was performed.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: Program error. The function was not performed because the user-specified area was not on a doubleword boundary.</td>
</tr>
<tr>
<td>08</td>
<td>Meaning: Program error. The function was not performed because the user supplied an invalid address.</td>
</tr>
<tr>
<td>10</td>
<td>Meaning: System error. The function was not performed because a machine check occurred.</td>
</tr>
<tr>
<td>14</td>
<td>Meaning: System error. The function was not performed because a program check occurred.</td>
</tr>
</tbody>
</table>

These return codes are passed to the error routine if it receives control.

Example 1

Place the value of the CPU timer in microseconds in location TIMELEFT.

CPUTIMER MIC,TIMELEFT

Example 2

Store the value of the CPU timer in time units in the location addressed by register 1.

CPUTIMER TU,(1)

Example 3

Store the value of the CPU timer in timer units in location TIMELEFT. If an error occurs, transfer control to the error routine labeled ERREXIT.

CPUTIMER ,TIMELEFT,ERRET=ERREXIT
Example 4

Place the value of the CPU timer in microseconds in the location addressed by register 1. If an error occurs, transfer control to the address in register 2.

CPUTIMER MIC,(1),ERRET=(2)
Chapter 33. CSRCESRV — Compress and expand data

Description

Use the CSRCESRV macro to compress data and restore the data to its original state when you need it. The CSRCESRV macro has three different services:

- Query (SERVICE=QUERY), to obtain the information your program needs to invoke data compression or data expansion
- Data compression (SERVICE=COMPRESS), to achieve reduced data volume
- Data expansion (SERVICE=EXPAND), to expand data previously compressed by the data compression service.

Before attempting to use the CSRCESRV macro, see “Using Data Compression and Expansion Services” in z/OS MVS Programming: Assembler Services Guide for a description of the data compression, expansion, and query services, and the conditions under which programs can exploit these services.

To invoke the CSRCESRV macro for either data compression or data expansion, first invoke CSRCESRV with SERVICE=QUERY. Follow these steps:

1. Load the general purpose registers (GPRs) with information required by SERVICE=QUERY.
2. Invoke the CSRCESRV macro with SERVICE=QUERY.
3. Load the GPRs with information required by SERVICE=COMPRESS (or SERVICE=EXPAND).
4. Invoke the CSRCESRV macro with SERVICE=COMPRESS (or SERVICE=EXPAND).
5. If all the input data has not been processed, continue to re-invoke the service until processing is complete.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB mode.
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 31-bit
ASC mode: Primary or access register (AR)
Interrupt status: Enabled or disabled for I/O and external interrupts
Locks: The caller may hold locks, but is not required to hold any.
Control parameters: None.

Programming requirements

None.

Restrictions

None.

Input register information for SERVICE=QUERY

Before issuing the CSRCESRV macro with the SERVICE=QUERY parameter, the caller must ensure that the following GPRs contain the specified information:
CSRCSERV macro

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The run length encoding algorithm. Specify either 0 or 1.</td>
</tr>
<tr>
<td>13</td>
<td>The 31-bit address of a standard 18-word save area. If your program is running in AR ASC mode, set AR 13 to specify the ALET to be used to qualify GPR 13.</td>
</tr>
</tbody>
</table>

Output register information for SERVICE=QUERY

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A value of 1 to indicate that the run length encoding algorithm will be used.</td>
</tr>
<tr>
<td>1</td>
<td>The length of the work area required by the algorithm. This value might be zero if the service does not require a work area.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Return code.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Input register information for SERVICE=COMPRESS

Before issuing the CSRCSERV macro with the SERVICE=COMPRESS parameter, the caller must ensure that the following GPRs contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The 31-bit address of a work area, if one is needed. The value returned in GPR 1, when you issue CSRCSERV with SERVICE=QUERY, indicates the size of the required work area. If your program is running in AR ASC mode, set AR 1 to specify the ALET to be used to qualify the GPR.</td>
</tr>
<tr>
<td>2</td>
<td>The 31-bit address of the uncompressed input data block. If your program is running in AR ASC mode, set AR 2 to specify the ALET to be used to qualify the GPR.</td>
</tr>
<tr>
<td>3</td>
<td>The length of the uncompressed input data block.</td>
</tr>
<tr>
<td>4</td>
<td>The 31-bit address of the output data block to hold the compressed data. If your program is running in AR ASC mode, set AR 4 to specify the ALET to be used to qualify the GPR.</td>
</tr>
<tr>
<td>5</td>
<td>The length of the output data block.</td>
</tr>
<tr>
<td>13</td>
<td>The 31-bit address of a standard 18-word save area. If your program is running in AR ASC mode, set AR 13 to specify the ALET to be used to qualify the GPR.</td>
</tr>
</tbody>
</table>

Output register information for SERVICE=COMPRESS

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Unchanged</td>
</tr>
<tr>
<td>2</td>
<td>The 31-bit address of the byte following the last input byte processed</td>
</tr>
<tr>
<td>3</td>
<td>The number of bytes of uncompressed data not processed</td>
</tr>
<tr>
<td>4</td>
<td>The 31-bit address of the byte following the last output byte</td>
</tr>
</tbody>
</table>
CSRCESRV macro

5  The number of bytes in the output data block into which output was not stored
6-13  Unchanged
14  Used as a work register by the system
15  Return code

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Input register information for SERVICE=EXPAND

Before issuing the CSRCESRV macro with the SERVICE=EXPAND parameter, the caller must ensure that the following GPRs contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The 31-bit address of a work area, if one is needed. The value returned in GPR 1, when you issue CSRCESRV with SERVICE=QUERY, indicates the size of the required work area. If your program is running in AR ASC mode, set AR 1 to specify the ALET to be used to qualify the GPR.</td>
</tr>
<tr>
<td>2</td>
<td>The 31-bit address of the compressed input data block. If your program is running in AR ASC mode, set AR 2 to specify the ALET to be used to qualify the GPR.</td>
</tr>
<tr>
<td>3</td>
<td>The length of the compressed input data block.</td>
</tr>
<tr>
<td>4</td>
<td>The 31-bit address of the output data block to hold the expanded data. If your program is running in AR ASC mode, set AR 4 to specify the ALET to be used to qualify the GPR.</td>
</tr>
<tr>
<td>5</td>
<td>The length of the output data block.</td>
</tr>
<tr>
<td>13</td>
<td>The 31-bit address of a standard 18-word save area. If your program is running in AR ASC mode, set AR 13 to specify the ALET to be used to qualify the GPR.</td>
</tr>
</tbody>
</table>

Output register information for SERVICE=EXPAND

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Unchanged</td>
</tr>
<tr>
<td>2</td>
<td>The 31-bit address of the byte following the last input byte processed</td>
</tr>
<tr>
<td>3</td>
<td>The number of bytes of compressed data not processed</td>
</tr>
<tr>
<td>4</td>
<td>The 31-bit address of the byte following the last output byte</td>
</tr>
<tr>
<td>5</td>
<td>The number of bytes in the output data block into which output was not stored</td>
</tr>
<tr>
<td>6-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
**CSRCESRV macro**

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 CSRCESRV macro is written as follows:

```
name name
b
CSRCESRV
b

,VECTOR=(reg) reg: register (2) - (12).
```

**Parameters**

The parameters are explained as follows:

- **SERVICE=QUERY**
  - Invokes the query service, which determines the following:
    - Whether data compression is supported by the system currently installed
    - The size of the work area required by the compression or expansion service.
  - You need the above information before you can invoke the macro with SERVICE=COMPRESS or SERVICE=EXPAND.

- **SERVICE=COMPRESS**
  - Invokes the data compression service, which compresses a given block of data, and stores the compressed data in an output area. You must obtain storage for this output area, and for a work area if SERVICE=QUERY returns a nonzero value in GPR 1. SERVICE=COMPRESS will compress as much of the input data as possible. It returns to the caller when either of the following has occurred:
    - It has compressed all the input data
    - It has completely filled the output area with the compressed data.
SERVICE=EXPAND invokes the data expansion service, which expands data that was previously compressed by the data compression service, and stores that data in its original form in an output area. You must obtain storage for this output area, and for a work area if SERVICE=QUERY returns a nonzero value in GPR 1. SERVICE=EXPAND will expand as much of the input data as possible. It returns to the caller when either of the following has occurred:
- It has expanded all the input data
- It has completely filled the output area with the expanded data.

VECTOR=(reg)
reg is the GPR that your program loads with the entry point address of the CSRCEXA load module. This load module resides in SYS1.MIGLIB.

ABEND codes
None.

Return and reason codes
When control is returned from CSRCESRV, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: The requested algorithm is supported. This means that both compression and expansion are supported.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: The requested algorithm is supported only for data expansion.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>0C</td>
<td>Meaning: Program error. The requested algorithm is not supported by this level of MVS.</td>
</tr>
<tr>
<td></td>
<td>Action: Specify the appropriate input value and rerun the program.</td>
</tr>
<tr>
<td>10</td>
<td>Meaning: Program error. The algorithm number was negative.</td>
</tr>
<tr>
<td></td>
<td>Action: Specify the appropriate input value and rerun the program.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Meaning: All input data was compressed.</td>
</tr>
<tr>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>Meaning: Program error. Not all input data was compressed because the output area was too small.</td>
</tr>
<tr>
<td></td>
<td>Action: Examine the information returned in the GPRs. Either make a follow-up request to have the rest of the uncompressed data processed, or issue the macro with a larger output area.</td>
</tr>
<tr>
<td>0C</td>
<td>Meaning: Program error. Either the input or output length was negative.</td>
</tr>
<tr>
<td></td>
<td>Action: Inspect the contents of the GPRs to determine which value is in error. Specify the appropriate input values and rerun the program.</td>
</tr>
</tbody>
</table>
### Table 9. Return Codes for SERVICE=EXPAND

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | **Meaning:** All input data was expanded.  
  **Action:** None. |
| 04                      | **Meaning:** Program error. Not all input data was expanded because the output area was too small.  
  **Action:** Examine the information returned in the GPRs. Either make a follow-up request to have the rest of the compressed data processed or issue the macro with a larger output area. |
| 08                      | **Meaning:** Program error. The data was not expanded because it was compressed by an up-level version of the data compression service, using an algorithm not understood by this version of the data expansion service.  
  **Action:** Check to see if all the input values were correct. Ensure that the input data was compressed by the appropriate data compression service and that the appropriate data expansion service was invoked. If the problem persists, record the return code and supply it to the appropriate IBM support personnel. |
| 0C                      | **Meaning:** Program error. Either the input or output length was negative. Inspect the register contents to make this distinction.  
  **Action:** Specify the appropriate input values and rerun the program. |
| 10                      | **Meaning:** Program error. Not all the data was expanded because the input data was not compressed by the data compression service.  
  **Action:** Check to see if all the input values were correct. Ensure that the input data was compressed by the appropriate data compression service and that the appropriate data expansion service was invoked. If the problem persists, record the return code and supply it to the appropriate IBM support personnel. |
Chapter 34. CSRCMPSC — Compress and expand data

Description

The CSRCMPSC macro performs the following functions:
- Compresses data
- Expands previously-compressed data

Environment

The requirements for the caller are:

Minimum authorization: Problem state, PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
Amode: 31-bit
ASC mode: Primary or access register (AR)
Interrupt status: Enabled or disabled for I/O and external interrupts
Locks: The caller may hold the local lock of the primary address space and may additionally hold the CMS lock. The caller may hold the CPU lock. No locks are required.

Control parameters: The CSRYCMPS area, and the dictionary, source area, and target area pointed to by the CSRYCMPS area can all be in the primary address space or, for AR-mode callers, in an address/data space addressable through an ALET. The dictionary and source areas are assumed to be in the same space. In the CSRYCMPS area, the fields that designate the ALETs of the dictionary, source, and target areas should be set to zero by primary mode callers. All parameters may reside in storage above 16 megabytes.

Programming requirements

Before running the CSRCMPSC macro, the program must provide:
- A CSRYCMPS area, using the CSRYCMPS mapping macro. The area is specified in the CBLOCK parameter of the CSRCMPSC macro.
- Dictionaries for the compress and expand services, using the CSRYCMPD mapping macro. The CSRYCMPS area gives the address of the dictionaries.
- A source area, which contains the data to be compressed or expanded. The CSRYCMPS area gives the address of the source area.
- A target area, which contains the data after the service has compressed or expanded it. The CSRYCMPS area gives the address of the target area.


Restrictions

None.

Input register information

Before issuing the CSRCMPSC macro, the caller must ensure that general purpose register (GPR) 13 contains the address of a standard 72-byte save area in the primary address space.
CSRCMPSC macro

Before issuing the CSRCMPSC macro, the caller does not have to place any information into any access register (AR), unless running in AR ASC mode. In this case, the caller must ensure that the following ARs contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>0 which designates the primary address space</td>
</tr>
</tbody>
</table>

If the caller is in AR mode and specifies CBLOCK=(n), or if the caller is in primary mode and specifies CBLOCK=(1), the caller must ensure that the following ARs contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The ALET with which the system is to access the CSRYCMPS area. For primary mode callers, the ALET should be 0.</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

The standard form of CSRCMPSC is written as follows:

```

name name: Symbol. Begin name in column 1.

b

CSRCMPSC

b

CBLOCK=comp block: RS-type address, or register (1) - (12).

,RETCODE=rc: RS-type address, or register (2) - (12).

Default: No return code processing.
```

---

188  z/OS V1R13.0 MVS Assembler Services Reference ABE-HSP
Parameters

The parameters are explained as follows:

**CBLOCK=**comp block

Specifies the address of the CSRYCMPS area. If register notation is used, the register contains the address of the area. The CSRYCMPS area contains the parameter information for the macro. The area is mapped by DSECT CMPSC in mapping macro CSRYCMPS; see `z/OS MVS Data Areas` in `z/OS Internet Library` at [http://www.ibm.com/systems/z/os/zos/bkserv/](http://www.ibm.com/systems/z/os/zos/bkserv/) for the CSRYCMPS macro.

**RETCODE=**rc

Specifies the fullword location where the system is to store the return code. If register notation is used, the system stores the return code into the register. In either case, the system stores the return code in GPR 15.

Abend codes

The program issuing CSRCMPSC may receive the listed abend codes. See `z/OS MVS System Codes`.

**0C4**

The program may get this completion code if the system cannot access the CSRYCMPS area, source area, target area, or dictionary.

**0C6**

The program may get this completion code if the CMPSC_SYMSIZE field in the CSRYCMPS area does not contain 1-5.

This completion code is received only if bit CVTCMPSC in mapping macro CVT is on.

**0C7**

The dictionary is built incorrectly. The program may receive this completion code in the following circumstances:

- If the length of a string to be represented by a single compression symbol, encountered during a compression operation, exceeds 260 characters.
- If a dictionary entry has more than 260 total child characters.
- If the child count in a dictionary entry indicates more than 6 child characters.
- If the number of extension characters for a dictionary entry with 0 or 1 child characters exceeds 4.
- If a sibling descriptor dictionary entry has a sibling count of 0.
- If expansion of a compression symbol uses more than 260 characters.
- If expansion of a compression symbol uses more than 127 dictionary entries.

In all these cases, the programmer needs to fix the dictionary.

This completion code is received only if bit CVTCMPSC in mapping macro CVT is on.
### Return and reason codes

When the CSRCMPSC macro returns control to the program, the RETCODE parameter fullword and GPR 15 contain one of the following hexadecimal return codes.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** Successful completion. Source operand was completely processed.  
Action: None. |
| 4                       | **Meaning:** Source operand was not completely processed. No room is left in the target operand.  
Action: Specify a larger target operand. Or provide another area for the target operand. Issue the macro again to resume processing of the operation. |
| 10                      | **Meaning:** Program error. A field in the CSRYCMPS area does not contain a value.  
Action: Provide values in the CMPSC_DICTADDR, CMPSC_TARGETADDR, and CMPSC_SOURCEADDR fields. |
| 14                      | **Meaning:** Program error. The symbol size in the CSRYCMPS area does not have a value of 1 through 5.  
Action: Provide a value of 1 through 5 in the CMPSC_SYMSIZE field. |
| 18                      | **Meaning:** The target area for compression or the source area for expansion is not large enough to hold even one compression symbol. The length of the area is specified in the CSRYCMPS area.  
Action: If this result is expected, no action is required. Otherwise, provide a larger value in the CMPSC_TARGETLEN field for compression or the CMPSC_SOURCELEN field for expansion. |
| 1C                      | **Meaning:** Program error. The length of the string represented by a single compression symbol exceeds the limit of 260 bytes.  
Action: Fix the dictionary. |
| 20                      | **Meaning:** Program error. The number of child characters for a compression dictionary entry exceeds 260.  
Action: Fix the dictionary. |
| 24                      | **Meaning:** Program error. A compression dictionary entry indicates that it contains more than 6 child characters, not including sibling characters.  
Action: Fix the dictionary. |
| 28                      | **Meaning:** Program error. The number of extension characters for a compression dictionary entry with 0 or 1 child characters exceeds 4.  
Action: Fix the dictionary. |
| 2C                      | **Meaning:** Program error. A sibling descriptor compression dictionary entry has a count of 0.  
Action: Fix the dictionary. |
| 30                      | **Meaning:** Program error. Expansion of a compression symbol used more than 127 dictionary entries.  
Action: Fix the dictionary. |
CSRCMPSC macro

Example 1

Compress a data area. Note that the expansion dictionary must immediately follow the compression dictionary, and both must be aligned on page boundaries.

> LA 2,MYCBLOCK Get address of parm
> USING CMPSC,2
> XC CMPSC(CMPSC_LEN),CMPSC Clear block
> OI CMPSC_FLAGS_BYTE2,CMPSC_SYMSIZE_5 Set size
> * Symbol size is 5+8. Dictionary has
> * 2**(5+8) entries
> L 3,DICTADDR
> ST 3,CMPSC_DICTADDR Set dictionary address
> L 3,COMPADDR
> ST 3,CMPSC_TARGETADDR Set compression area
> L 3,COMPLEN
> ST 3,CMPSC_TARGETLEN Set compression length
> L 3,EXPADDR
> ST 3,CMPSC_SOURCEADDR Set expansion area
> L 3,EXPLEN
> ST 3,CMPSC_SOURCELEN Set expansion length
> LA 3,WORKAREA
> ST 3,CMPSC_WORKAREAADDR Set work area address
> CSRCMPSC CBLOCK=CMPSC
> DROP 2

DS 0F Align parameter on word boundary
MYCBLOCK DS (CMPSC_LEN)CL1 CBLOCK parameter
COMPADDR DS A Output compression area
COMPLEN DS F Length of compression area
EXPADDR DS A Input expansion area
EXPLEN DS F Length of expansion area
DICTADDR DS A Address of compression dictionary
DS 0D Doubleword align work area
WORKAREA DS CL192 Work area
CSRCMPSC ,

Example 2

Expand a data area. Note that the expansion dictionary must be aligned on a page boundary.

> LA 2,MYCBLOCK Get address of parm
> USING CMPSC,2
> XC CMPSC(CMPSC_LEN),CMPSC Clear block
> OI CMPSC_FLAGS_BYTE2,CMPSC_SYMSIZE_5 Set size
> * Symbol size is 5+8. Dictionary has
> * 2**(5+8) entries
> OI CMPSC_FLAGS_BYTE2,CMPSC_EXPAND Do expansion
> L 3,DICTADDR
> ST 3,CMPSC_DICTADDR Set dictionary address
> L 3,EXPADDR
> ST 3,CMPSC_TARGETADDR Set expansion area
> L 3,EXPLEN
> ST 3,CMPSC_TARGETLEN Set expansion length
> L 3,COMPADDR
> ST 3,CMPSC_SOURCEADDR Set compression area
> L 3,COMPLEN
> ST 3,CMPSC_SOURCELEN Set compression length
> LA 3,WORKAREA
> ST 3,CMPSC_WORKAREAADDR Set work area address
> CSRCMPSC CBLOCK=CMPSC
> DROP 2


**CSRCMPSC macro**

``` assembler
DS OF 0F   Align parameter on word boundary
MYCBLOCK DS (CMPSC_LEN)CL1 CBLOCK Parameter
EXPADDR DS A  Output expansion area
EXPLEN DS F  Length of expansion area
COMPADDR DS A  Input compression area
COMPLEN DS F  Length of compression area
EDICTADDR DS A  Address of expansion dictionary
DS 0D Doubleword align work area
WORKAREA DS CL192 Work area
CSRCMPSC ,
```

**Example 3**

When using register notation in the CBLOCK parameter, the program must place both the address and ALET into a GPR/AR pair. This is true whether you are running in AR or primary ASC mode.

``` assembler
LAE 2,MYCBLOCK Set address and ALET
CSRCMPSC CBLOCK=(2) Issue operation
```

---

**z/OS V1R13.0 MVS Assembler Services Reference ABE-HSP**
Chapter 35. CSRC4ACT — Activate previously connected storage

Description

Call the CSRC4ACT cell pool service to activate the extent cell storage for allocation. You must specify which extent you want to activate.

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4ACT so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:

```
CSRC4_ANCHOR_LENGTH    EQU 64
```

* Base length of the cell pool extent data area:

```
CSRC4_EXTENT_BASE      EQU 192
```

* Length of the user-supplied pool name:

```
CSRC4_POOL_NAME_LEN    EQU 8
```
CSRC4ACT callable service

Restrictions

None.

Input register information
Before calling the CSRC4ACT service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

CALL CSRC4ACT
,(cnt1_alet
,anchor_addr
,extent_num
,return_code)

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

,(cntl_alet
  Specifies the fullword variable containing the ALET identifying the location of the anchor and extents. Initialize the ALET to 0 if your program is in AR mode and
CSRC4ACT callable service

the anchor and extents are in the primary address space. If your program is
running in primary ASC mode, the value is ignored, but you must code the
parameter anyway.

anchor_addr
Identifies the doubleword variable containing the address of the 64-byte anchor.

extent_num
Identifies the fullword variable containing the number of the extent to be
connected. The extent number must be within the range 0 to 65536.

return_code
When CSRC4ACT completes, the fullword variable specified by return_code
contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRC4ACT service returns control to your program, GPR 15 (and
return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful. Action: None.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid. Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>Meaning: Program error. The extent number is not valid. Action: Make sure the extent number is within the range 0 to 65536.</td>
</tr>
<tr>
<td>34</td>
<td>52</td>
<td>Meaning: Program error. The extent is in the incorrect state. Action: Check to see if your program passed the wrong extent number. Make sure the extent is not already in an active state (that is, it has not been activated through CSRC4ACT or CSRC4EXP).</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>Meaning: Program or system error. An extent chain was broken. Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>68</td>
<td>104</td>
<td>Meaning: Program or system error. An extent chain is circular. Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
**CSRC4ACT callable service**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 6C                      | 108                 | **Meaning**: Program or system error. An extent could not be found.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
Chapter 36. CSRC4BLD — Build a cell pool and initialize an anchor

Description

Call the CSRC4BLD cell pool service to format a 64-byte area for the cell pool anchor. You must first have acquired the storage for the anchor. You can call this service only once for a given cell pool.

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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:** All parameters must reside in a single address or data space, and must be addressable by the caller. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call the CSRC4BLD service so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

```plaintext
* Length of the cell pool anchor data area:
*  
  CSRC4_ANCHOR_LENGTH   EQU   64

* Base length of the cell pool extent data area:
*  
  CSRC4_EXTENT_BASE     EQU   192

* Length of the user-supplied pool name:
*  
  CSRC4_POOL_NAME_LEN   EQU   8
```

© Copyright IBM Corp. 1988, 2012
CSRC4BLD callable service

Restrictions
None.

Input register information
Before calling the CSRC4BLD service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```assembly
 CALL CSRC4BLD,
   (.cntl_alet,
    .anchor_addr,
    .user_name,
    .cell_size,
    .return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
.cntl_alet
```
Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR
CSRC4BLD callable service

mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

\textbf{anchor_addr}

Specifies the doubleword variable containing the address of the cell pool anchor.

\textbf{user_name}

Specifies the 8-byte variable containing the name you want the service to assign to the pool. There are no restrictions on the name.

\textbf{cell_size}

Specifies the doubleword variable containing the cell size in this pool. You can use any positive binary or hexadecimal number as the cell size.

\textbf{return_code}

When CSRC4BLD completes, the fullword variable specified by \textit{return_code} contains the return code.

\section*{ABEND codes}

None.

\section*{Return and reason codes}

When the CSRC4BLD service returns control to your program, GPR 15 (and \textit{return_code}) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | \textit{Meaning}: The operation was successful.  
\textit{Action}: None. |
| 18                      | 24                  | \textit{Meaning}: Program error. The anchor address is not valid.  
\textit{Action}: Check to see if your program passed the wrong anchor address. If the anchor is in a data space, make sure the anchor address is at least 63 bytes less than the address of the last byte of the data space. |
| 44                      | 68                  | \textit{Meaning}: Program error. The cell size is not valid: it cannot be negative or 0.  
\textit{Action}: Specify a positive value for the cell size. |
CSRC4BLD callable service
Chapter 37. CSRC4CON — Connect cell storage to an extent

Description

Call the CSRC4CON cell pool service to connect cell storage to the extent that you specify or to reuse a disconnected extent. The CSRC4EXP service returned the extent number. The extent must be in the disconnected state, which means that you have not called CSRC4ACT to activate this particular extent.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in a primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4CON so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
* CSRC4_ANCHOR_LENGTH EQU 64
*
* Base length of the cell pool extent data area:
* CSRC4_EXTENT_BASE EQU 192
*
* Length of the user-supplied pool name:
* CSRC4_POOL_NAME_LEN EQU 8
*

Restrictions

None.
CSRC4CON callable service

Input register information
Before calling the CSRC4CON service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
,(cntl_alet
,anchor_addr
,area_addr
,area_size
,extent_num
,return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
,(cntl_alet
```

Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, you must code this parameter, even though any value that you code is ignored.
CSRC4CON callable service

**anchor_addr**
Specifies the doubleword variable containing the address of the 64-byte anchor.

**area_addr**
Specifies the doubleword variable containing the starting address of the cell storage area. The starting address of this area must be consistent with any cell boundary requirements that you might have.

**area_size**
Specifies the doubleword variable containing the length of the cell storage area. CSRC4CON determines the number of cells that will fit in the area.

**extent_num**
Specifies the fullword variable containing the number of the extent to be connected. The extent number must be within the range 0 to 65536.

**return_code**
When CSRC4CON completes, the fullword variable specified by return_code contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRC4CON service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td><strong>Meaning:</strong> The operation was successful. <strong>Action:</strong> None.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td><strong>Meaning:</strong> Program error. The anchor address is not valid. <strong>Action:</strong> Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td><strong>Meaning:</strong> Program error. The extent number is not valid. <strong>Action:</strong> Specify the extent number within the range 0 to 65536.</td>
</tr>
<tr>
<td>34</td>
<td>52</td>
<td><strong>Meaning:</strong> Program error. You issued the services in the wrong order, or did not issue a necessary service. <strong>Action:</strong> Check to see if your program passed the wrong extent number. Make sure that the extent is in a disconnected state (that is, it has not been activated through CSRC4ACT or CSRC4EXP).</td>
</tr>
<tr>
<td>48</td>
<td>72</td>
<td><strong>Meaning:</strong> Program error. The cell area length is not valid. <strong>Action:</strong> Check the specified cell area length. It should not be less than the cell size.</td>
</tr>
</tbody>
</table>
## CSRC4CON callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4C                      | 76                  | **Meaning:** Program error. The service could not access the cell area address.  
**Action:** If the cell area is in a data space, make sure the cell area is completely within the data space. |
| 50                      | 80                  | **Meaning:** Program error. The cell area is too large.  
**Action:** Specify a larger extent size or a smaller cell area size. |
| 64                      | 100                 | **Meaning:** Program or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 6C                      | 108                 | **Meaning:** Program or system error. An extent could not be found.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
Chapter 38. CSRC4DAC — Deactivate an extent

Description

Call the CSRC4DAC cell pool service to deactivate a specific extent. Use this service to prepare the cell pool for contraction. You must specify which extent you want to deactivate.

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN.
- **AMODE:** 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4DAC so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area:**
  
  ```
  CSRC4_ANCHOR_LENGTH   EQU  64
  *
  *
  * Base length of the cell pool extent data area:
  *
  CSRC4_EXTENT_BASE     EQU  192
  *
  *
  * Length of the user-supplied pool name:
  *
  CSRC4_POOL_NAME_LEN   EQU  8
  ```
CSRC4DAC callable service

Restrictions
After calling CSRC4DAC, you can still free (or return) cells, but you cannot get (or allocate) any others for this extent.

Input register information
Before calling the CSRC4DAC service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

CALL CSRC4DAC

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

,.cntl_alet
   Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR
mode and the anchor and extents are in the primary address space. If your
program is running in primary ASC mode, the value is ignored, but you must
code the parameter anyway.

anchor_addr
  Specifies the doubleword variable containing the address of the 64-byte anchor.

extent_num
  Specifies the fullword variable containing the number of the extent that
  CSRC4DAC will deactivate. The extent number must be within the range 0 to
  65536.

return_code
  When CSRC4DAC completes, the fullword variable specified for return_code
  contains the return code.

ABEND codes
  None.

Return and reason codes
  When the CSRC4DAC service returns control to your program, GPR 15 (and
  return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Meaning: The last cell has been returned to an inactive extent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None required. However, you might take some action depending on your application.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>Meaning: Program error. The extent number is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Make sure the extent number is within the range 0 to 65536.</td>
</tr>
<tr>
<td>34</td>
<td>52</td>
<td>Meaning: Program error. You issued the services in the wrong order, or did not issue a necessary service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program passed the wrong extent number. Make sure that the extent is in active state before calling the service.</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>Meaning: Program error or system error. An extent chain was broken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
### CSRC4DAC callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 6C                      | 108                 | **Meaning**: Program error or system error. An extent could not be found.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
Chapter 39. CSRC4DIS — Disconnect the cell storage for an extent

Description

Call the CSRC4DIS cell pool service to disconnect cell storage for a specific extent.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 64-bit addressing mode.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4DIS so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you call CSRC4DIS, you must have returned all cells associated with the extent and have called CSRC4DAC to deactivate the extent.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  `CSRC4_ANCHOR_LENGTH` EQU 64
- Base length of the cell pool extent data area:
  `CSRC4_EXTENT_BASE` EQU 192
- Length of the user-supplied pool name:
  `CSRC4_POOL_NAME_LEN` EQU 8

© Copyright IBM Corp. 1988, 2012
CSRC4DIS callable service

Restrictions
None.

Input register information
Before calling the CSRC4DIS service, the caller does not have to place any
information into any register unless using it in register notation for a particular
parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs)
contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on
the CALL statement in the order shown.

CALL CSRC4DIS

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

,(cntl_alet
   .anchor_addr
   .extent_num
   .area_addr
   .area_size
   .return_code)
CSRC4DIS callable service

the anchor and extents. Initialize the ALET to 0 if your program is running in AR
mode and the anchor and extents are in the primary address space. If your
program is running in primary ASC mode, the value is ignored, but you must
code the parameter anyway.

,anchor_addr
Specifies the doubleword variable containing the address of the 64-byte anchor.

,extent_num
Specifies the fullword variable containing the number of the extent that
CSRC4DIS will disconnect. The extent number must be within the range 0 to
65536.

,area_addr
When CSRC4DIS completes, the doubleword variable specified by area_addr
contains the address of the disconnected storage area.

,area_size
When CSRC4DIS completes, the doubleword variable specified by area_size
contains the size of the disconnected area.

,return_code
When CSRC4DIS completes, the fullword variable specified by return_code
contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRC4DIS service returns control to your program, GPR 15 (and
return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>Meaning: Program error. The extent number is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Make sure the extent number is within the range 0 to 65536.</td>
</tr>
<tr>
<td>34</td>
<td>52</td>
<td>Meaning: Program error. You issued the services in the wrong order, or did not issue a necessary service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Call CSRC4DAC to deactivate the extent before calling CSRC4DIS to disconnect the cell storage for the extent.</td>
</tr>
</tbody>
</table>
### CSRC4DIS callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 38                      | 56                  | **Meaning:** Program error. The service cannot disconnect the extent because some cells are still allocated.  
**Action:** Return all the cells associated with the extent before calling CSRC4DIS to disconnect the cell storage for the extent. |
| 64                      | 100                 | **Meaning:** Program or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 6C                      | 108                 | **Meaning:** Program or system error. An extent could not be found.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
Chapter 40. CSRC4EXP — Expand a cell pool

Description

Call the CSRC4EXP cell pool service to:
- Add an extent to the cell pool
- Assign a number to the extent
- Optionally, establish a connection between the extent and cell storage
- Optionally, make the cell storage available for allocation.

Note: If you are reusing an extent, use CSRC4CON and CSRC4ACT instead of CSRC4EXP.

If you specify zero for the cell storage size, CSRC4EXP will add an extent to the cell pool, but will keep it in a disconnected state. When you specify the extent size, allow 192 bytes plus one byte per eight cells of cell storage. CSRC4EXP allocates cells contiguously, starting at the address you specify. If you specify zero for the area length, CSRC4EXP ignores the area address.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4EXP so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
* CSRC4_ANCHOR_LENGTH EQU 64
*
CSRC4EXP callable service

* Base length of the cell pool extent data area:
  *
  CSRC4_EXTENT_BASE EQU 192
  *
  *
* Length of the user-supplied pool name:
  *
  CSRC4_POOL_NAME_LEN EQU 8
  *
  *
Restrictions

None.

Input register information

Before calling the CSRC4EXP service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

\( \text{cntl}_alet \)
- Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

\( \text{anchor}_addr \)
- Specifies the doubleword variable containing the address of the 64-byte anchor.

\( \text{extent}_addr \)
- Specifies the doubleword variable containing the address of the extent.

\( \text{extent}_size \)
- Specifies the doubleword variable containing the size of the extent.

\( \text{area}_addr \)
- Specifies the doubleword variable containing the starting address of the cell storage area. The starting address of this area must be consistent with any boundary requirements that you might have.

\( \text{area}_size \)
- Specifies the doubleword variable containing the length (binary or hexadecimal) of the storage area for the cells.

\( \text{extent}_num \)
- When CSRC4EXP completes, the fullword variable specifying \( \text{extent}_num \) contains the number of the extent to be connected. You will use this number on subsequent CALLs.

\( \text{return}_code \)
- When CSRC4EXP completes, the fullword variable specifying \( \text{return}_code \) contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRC4EXP service returns control to your program, GPR 15 (and \( \text{return}_code \)) contains one of the following return codes:
## CSRC4EXP callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                   | **Meaning:** The operation was successful.  
**Action:** None. |
| 0C                      | 12                   | **Meaning:** Program error. There are too many extents in the cell pool.  
**Action:** Check to see if your program contains a logic error that caused the limit of 65536 extents per cell pool to be exceeded. If your program works as expected, consider using a larger cell pool. |
| 1C                      | 28                   | **Meaning:** Program error. The anchor address is not valid.  
**Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 28                      | 40                   | **Meaning:** Program error. The service could not use the extent address.  
**Action:** If the extent is in a data space, make sure the extent address is at least 128 bytes less than the address of the last byte of the data space. Also make sure the extent area does not overlap the anchor area. |
| 2C                      | 44                   | **Meaning:** Program error. The extent length is not valid.  
**Action:** Correct the extent length. It cannot be less than 129 bytes. |
| 48                      | 72                   | **Meaning:** Program error. The cell area length is not valid.  
**Action:** Correct the cell area length. The cell area size cannot be less than the cell size. |
| 4C                      | 76                   | **Meaning:** Program error. The service could not use the cell area address.  
**Action:** If the cell area is in a data space, make sure the cell area is completely within the data space. |
| 50                      | 80                   | **Meaning:** Program error. The cell area is too large.  
**Action:** Specify a larger extent size or a smaller cell area size. |
| 64                      | 100                  | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                  | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
### CSRC4EXP callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 70                      | 112                 | **Meaning:** Program error or system error. An anchor has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid the anchor area. |
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRC4EXP callable service
Chapter 41. CSRC4FRE — Return a cell to a cell pool

Description

Call the CSRC4FRE cell pool service to return an allocated cell to a cell pool. You must specify the address of the cell that you want to return. (The CSRC4FR1 and CSRC4FR2 service provides the same function with slightly enhanced performance. CSRC4FR2 is preferred over CSRC4FR1 when using multiple extents.)

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you CALL CSRC4FRE so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:

```
CSRC4_ANCHOR_LENGTH    EQU    64
```

* Base length of the cell pool extent data area:

```
CSRC4_EXTENT_BASE      EQU    192
```

* Length of the user-supplied pool name:

```
CSRC4_POOL_NAME_LEN    EQU    8
```
CSRC4FRE callable service

Restrictions
None.

Input register information
Before calling the CSRC4FRE service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRC4FRE
   (.cntl_alet,
    .anchor_addr,
    .cell_addr,
    .return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

`.cntl_alet`
Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR
mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

anchor_addr
Specifies the doubleword variable containing the address of the 64-byte anchor.

cell_addr
Specifies the doubleword variable containing the address of the cell that CSRC4FRE is to free.

return_code)
When CSRC4FRE completes, the fullword variable specified for return_code contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRC4FRE service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | Meaning: The operation was successful.  
Action: None. |
| 04                      | 04                  | Meaning: The last cell has been returned to an inactive extent.  
Action: None required. However, you might take some action depending on your application. |
| 1C                      | 28                  | Meaning: Program error. The anchor address is not valid.  
Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | Meaning: Program error. The cell address is not valid.  
Action: Investigate the following possible causes:  
• The input cell address does not point to the beginning of a cell.  
• The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | Meaning: Program error. Either you have already returned the cell or you never allocated it.  
Action: Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | Meaning: Program error or system error. An extent chain was broken.  
Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
## CSRC4FRE callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell. |
Chapter 42. CSRC4FR1 — Return a cell to a cell pool

Description

Call the CSRC4FR1 cell pool service to return an allocated cell to a cell pool. You must specify the address of the cell that you want to return. (The CSRC4FRE service provides the same function but slightly slower performance. CSRC4FR2 is preferred over CSRC4FR1 when using multiple extents.)

Environment

The requirements for the caller are:

| Minimum authorization:       | Problem state with PSW key 8-15          |
| Dispatchable unit mode:      | Task or SRB                               |
| Cross memory mode:           | Any PASN, any HASN, any SASN              |
| AMODE:                       | 64-bit addressing mode. All input addresses must be valid 64-bit addresses. |
| ASC mode:                    | Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.) |
| 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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB. |

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you CALL CSRC4FR1 so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:

  CSRC4_ANCHOR_LENGTH EQU 64

* Base length of the cell pool extent data area:

  CSRC4_EXTENT_BASE EQU 192

* Length of the user-supplied pool name:

  CSRC4_POOL_NAME_LEN EQU 8

* Length of the user-supplied savearea:
CSRC4FR1 callable service

Restrictions
None.

Input register information
Before calling the CSRC4FR1 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```assembly
call csrC4fr1, (cntl_alet, anchor_addr, cell_addr, return_code, save_area)
```
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

\(,\text{cnt1_alet}\)
  Specifies the fullword variable containing the ALET that identifies the location of
  the anchor and extents. Initialize the ALET to 0 if your program is running in AR
  mode and the anchor and extents are in the primary address space. If your
  program is running in primary ASC mode, the value is ignored, but you must
  code the parameter anyway.

\(,\text{anchor_addr}\)
  Specifies the doubleword variable containing the address of the 64-byte anchor.

\(,\text{cell_addr}\)
  Specifies the doubleword variable containing the address of the cell that
  CSRC4FR1 is to free.

\(,\text{return_code}\)
  When CSRC4FR1 completes, the fullword variable specified for \text{return_code}
  contains the return code.

\(,\text{save_area}\)
  Specifies a 216-byte save area. The system does not change the first 8 bytes
  of this area.

ABEND codes

None.

Return and reason codes

When the CSRC4FR1 service returns control to your program, GPR 15 (and
\text{return_code}) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful. Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Meaning: The last cell has been returned to an inactive extent. Action: None required. However, you might take some action depending on your application.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid. Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>54</td>
<td>84</td>
<td>Meaning: Program error. The cell address is not valid. Action: Investigate the following possible causes: • The input cell address does not point to the beginning of a cell. • The cell is not in the cell pool specified by the anchor address.</td>
</tr>
</tbody>
</table>
### CSRC4FR1 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 58                      | 88                  | **Meaning:** Program error. Either you have already returned the cell or you never allocated it.  
**Action:** Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell.
Chapter 43. CSRC4FR2 — Return a cell to a cell pool

Description

Call the CSRC4FR2 cell pool service to return an allocated cell to a cell pool. You must specify the address of the cell that you want to return. (The CSRC4FRE service provides the same function but slightly slower performance. CSRC4FR2 is preferred over CSRC4FR1 when using multiple extents, as CSRC4FR2 has an additional input parameter for the address of the extent containing the cell to be freed.)

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you CALL CSRC4FR2 so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
  * CSRC4_ANCHOR_LENGTH EQU 64
  *
  *
  * Base length of the cell pool extent data area:
  * CSRC4_EXTENT_BASE EQU 192
  *
  *
  * Length of the user-supplied pool name:
  * CSRC4_POOL_NAME_LEN EQU 8
  *
  *
CSRC4FR2 callable service

* Length of the user-supplied savearea:

* CSRC4_SAVEAREA_LEN EQU 216
* *

Restrictions
None.

Input register information
Before calling the CSRC4FR2 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

CALL CSRC4FR2 *(cntl_alet, anchor_addr, cell_addr, return_code, save_area), extent_addr
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

, (cntl_alet
  Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

, anchor_addr
  Specifies the doubleword variable containing the address of the 64-byte anchor.

,cell_addr
  Specifies the doubleword variable containing the address of the cell that CSRC4FR2 is to free.

,return_code
  When CSRC4FR2 completes, the fullword variable specified for return_code contains the return code.

,save_area)
  Specifies a 216-byte save area. The system does not change the first 8 bytes of this area.

,extent_addr
  Specifies the doubleword variable containing the address of the extent that contains the cell that CSRC4FR2 is to free. This address was returned with the cell address in the extent_addr parameter on a previous CSRC4GT2 call.

ABEND codes

None.

Return and reason codes

When the CSRC4FR2 service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning**: The operation was successful.  
Action: None. |
| 04                      | 04                  | **Meaning**: The last cell has been returned to an inactive extent.  
Action: None required. However, you might take some action depending on your application. |
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
## CSRC4FR2 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 54                      | 84                  | **Meaning**: Program error. The cell address is not valid.  
**Action**: Investigate the following possible causes:  
• The input cell address does not point to the beginning of a cell.  
• The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | **Meaning**: Program error. Either you have already returned the cell or you never allocated it.  
**Action**: Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning**: Program error or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell. |
Chapter 44. CSRC4GET — Allocate a cell from a cell pool

Description

Call the CSRC4GET cell pool service to allocate a cell from the cell pool. CSRC4GET allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to the highest cell address. CSRC4GET passes back to the calling program the address of the cell it allocated but does not clear the cell storage to binary zeros. (The CSRC4GT1 and CSRC4GT2 services provide the same function with slightly enhanced performance. CSRC4GT2 is preferred over CSRC4GT1 when using multiple extents.)

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4GET so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area**
  - CSRC4_ANCHOR_LENGTH EQU 64
- **Base length of the cell pool extent data area**
  - CSRC4_EXTENT_BASE EQU 192
- **Length of the user-supplied pool name**
CSRC4GET callable service

Restrictions
None.

Input register information
Before calling the CSRC4GET service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
| 15       | Return code in low 32 bits. High 32 bits are used as a work area by the system.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRC4GET
    (cntl_alet
     .anchor_addr
     .cell_addr
     .return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:
CSRC4GET callable service

\( \text{cnt1_alet} \)
Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

\( \text{anchor_addr} \)
Specifies the doubleword variable containing the address of the 64-byte anchor.

\( \text{cell_addr} \)
When CSRC4GET completes, the doubleword variable specified by \( \text{cell_addr} \) contains the address of the cell that CSRC4GET allocated.

\( \text{return_code} \)
When CSRC4GET completes, the fullword variable specified by \( \text{return_code} \) contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRC4GET service returns control to your program, GPR 15 (and \( \text{return_code} \)) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>Meaning: Program error. There were no available cells in the pool. More than one program could be using the cell pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Retry the request one or more times. If the problem persists, consider freeing existing cells or adding new cells to the cell pool, or both.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>Meaning: Program or system error. An extent chain was broken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>68</td>
<td>104</td>
<td>Meaning: Program or system error. An extent chain is circular.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
CSRC4GET callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 74                      | 116                 | **Meaning:** Program or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 45. CSRC4GT1 — Allocate a cell from a cell pool

Description

Call the CSRC4GT1 cell pool service to allocate a cell from the cell pool. CSRC4GT1 allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to the highest cell address. CSRC4GT1 passes back to the calling program the address of the cell it allocated but does not clear the cell storage to binary zeros. (The CSRC4GET service provides the same function but slightly slower performance. CSRC4GT2 is preferred over CSRC4GT1 when using multiple extents.)

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4GT1 so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

- * Length of the cell pool anchor data area:
  *  
  CSRC4_ANCHOR_LENGTH EQU 64
  *
  *
- * Base length of the cell pool extent data area:
  *  
  CSRC4_EXTENT_BASE EQU 192
  *
  *
- * Length of the user-supplied pool name:
  *  
  CSRC4_POOL_NAME_LEN EQU 8
  *
  *
CSRC4GT1 callable service

Restrictions

None.

Input register information

Before calling the CSRC4GT1 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRC4GT1
          ,cntl_alet
          ,anchor_addr
          ,cell_addr
          ,return_code
          ,save_area)
```
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

\( \text{cntl}_\text{alet} \)
Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

\( \text{anchor}_\text{addr} \)
Specifies the doubleword variable containing the address of the 64-byte anchor.

\( \text{cell}_\text{addr} \)
When CSRC4GT1 completes, the doubleword variable specified by \( \text{cell}_\text{addr} \) contains the address of the cell that CSRC4GT1 allocated.

\( \text{return}_\text{code} \)
When CSRC4GT1 completes, the fullword variable specified by \( \text{return}_\text{code} \) contains the return code.

\( \text{save}_\text{area} \)
Specifies a 216-byte save area. The system does not change the first 8 bytes of this area.

ABEND codes

None.

Return and reason codes

When the CSRC4GT1 service returns control to your program, GPR 15 (and \( \text{return}_\text{code} \)) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning**: The operation was successful.  
**Action**: None. |
| 08                      | 08                  | **Meaning**: Program error. There were no available cells in the pool. More than one program could be using the cell pool.  
**Action**: Retry the request one or more times. If the problem persists, consider freeing existing cells or adding new cells to the cell pool, or both. |
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
**Action**: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 64                      | 100                 | **Meaning**: Program or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
## CSRC4GT1 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 68                      | 104                 | **Meaning**: Program or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning**: Program or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 46. CSRC4GT2 — Allocate a cell from a cell pool

Description

Call the CSRC4GT2 cell pool service to allocate a cell from the cell pool. CSRC4GT2 allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to the highest cell address. CSRC4GT2 passes back to the calling program the address of the cell it allocated but does not clear the cell storage to binary zeros. (The CSRC4GET service provides the same function but slightly slower performance. CSRC4GT2 is preferred over CSRC4GT1 when using multiple extents, as CSRC4GT2 has an additional output parameter to return the address of the extent containing the obtained cell.)

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4GT2 so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area:**
  - `CSRC4_ANCHOR_LENGTH` EQU 64
- **Base length of the cell pool extent data area:**
  - `CSRC4_EXTENT_BASE` EQU 192
- **Length of the user-supplied pool name:**
  - `CSRC4_POOL_NAME_LEN` EQU 8
Restrictions

None.

Input register information

Before calling the CSRC4GT2 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
,(cntl_alet
,anchor_addr
,cell_addr
,return_code
,save_area)
,extent_addr
```
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

, (cnt1_alet
   Specifies the fullword variable containing the ALET that identifies the location of
   the anchor and extents. Initialize the ALET to 0 if your program is running in AR
   mode and the anchor and extents are in the primary address space. If your
   program is running in primary ASC mode, the value is ignored, but you must
   code the parameter anyway.

,anchor_addr
   Specifies the doubleword variable containing the address of the 64-byte anchor.

,cell_addr
   When CSRC4GT2 completes, the doubleword variable specified by cell_addr
   contains the address of the cell that CSRC4GT2 allocated.

,return_code
   When CSRC4GT2 completes, the fullword variable specified by return_code
   contains the return code.

,save_area)
   Specifies a 216-byte save area. The system does not change the first 8 bytes
   of this area.

,extent_addr
   Specifies the variable that is to contain the address of the extent containing the
   obtained cell. This address should be provided in the extent_addr parameter
   when using CSRC4FR2 to free the cell.

ABEND codes

None.

Return and reason codes

When the CSRC4GT2 service returns control to your program, GPR 15 (and
return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful. Action: None.</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>Meaning: Program error. There were no available cells in the pool. More than one program could be using the cell pool. Action: Retry the request one or more times. If the problem persists, consider freeing existing cells or adding new cells to the cell pool, or both.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid. Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
</tbody>
</table>
### CSRC4GT2 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 64                      | 100                 | **Meaning:** Program or system error. An extent chain was broken.  
                        |                     | **Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program or system error. An extent chain is circular.  
                        |                     | **Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning:** Program or system error. An extent has been overlaid.  
                        |                     | **Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 47. CSRC4QCL — Query a cell

Description
Call the CSRC4QCL cell pool service to receive status information about a specified cell in a cell pool. CSRC4QCL reports whether the cell is free or allocated, and returns the number of the extent associated with the cell. CSRC4QCL does not prevent other programs from changing the pool during or after a query. CSRC4QCL returns the status as it was at the time you issued the CALL.

Environment
The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements
If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4QCL so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

```assembly
* Length of the cell pool anchor data area:
* CSRC4_ANCHOR_LENGTH   EQU   64
*
* Base length of the cell pool extent data area:
* CSRC4_EXTENT_BASE      EQU   192
*
* Length of the user-supplied pool name:
* CSRC4_POOL_NAME_LEN    EQU   8
*
```

© Copyright IBM Corp. 1988, 2012
CSRC4QCL callable service

Restrictions
None.

Input register information
Before calling the CSRC4QCL service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRC4QCL
,(cnt1_alet
,anchor_addr
,cell_addr
,cell_avail
,extent_num
,return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
,(cnt1_alet
   Specifies the fullword variable containing the ALET that identifies the location of
```
the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

,anchor_addr
   Specifies the doubleword variable containing the address of the 64-byte anchor.

,cell_addr
   Specifies the doubleword variable containing the address of the cell the service will query.

,cell_avail
   When CSRC4QCL completes, the doubleword variable specified for cell_avail contains one of the following values. These indicate the status of the specified cell at the time you issued the CALL.
   0   Cell available
   1   Cell allocated

,extent_num
   When CSRC4QCL completes, the fullword variable specified for extent_num contains the number of the extent that contains the specified cell.

,return_code
   When CSRC4QCL completes, the fullword variable specified for return_code contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRC4QCL service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if the program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>54</td>
<td>84</td>
<td>Meaning: Program error. The cell address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Investigate the following possible causes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The input cell address does not point to the beginning of a cell</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The cell is not in the cell pool specified by the anchor address.</td>
</tr>
</tbody>
</table>
### CSRC4QCL callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 64                      | 100                 | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 48. CSRC4QEX — Query a cell pool extent

Description
Call the CSRC4QEX cell pool service to receive status information about a specified extent.

CSRC4QEX does not prevent other programs from changing the pool during or after a query. CSRC4QEX returns the status as it was at the time you issued the CALL.

Environment
The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. Control parameters must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements
If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4QEX so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
  * CSRC4_ANCHOR_LENGTH EQU 64

* * * Base length of the cell pool extent data area:
  * CSRC4_EXTENT_BASE EQU 192

* * *

* Length of the user-supplied pool name:
Restrictions

None.

Input register information

Before calling the CSRC4QEX service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```assembly
,(cnt1_alet,
.anchor_addr,
.extent_num,
.status,
.extent_addr,
.extent_len,
.area_addr,
.area_size,
.total_cells,
.avail_cells,
.return_code)
CALL CSRC4QEX
```
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

\texttt{cnt1\_alet}

Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

\texttt{anchor\_addr}

Specifies the doubleword variable containing the address of the 64-byte anchor.

\texttt{extent\_num}

Specifies the fullword variable containing the number of the extent the service will query.

\texttt{status}

When CSRC4QEX completes, the doubleword variable specified for \texttt{status} contains one of the following decimal numbers. These indicate the status of the extent at the time of the CALL.

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Disconnected and inactive</td>
</tr>
<tr>
<td>2</td>
<td>Connect in progress</td>
</tr>
<tr>
<td>3</td>
<td>Connected and inactive</td>
</tr>
<tr>
<td>4</td>
<td>Connected and active</td>
</tr>
<tr>
<td>5</td>
<td>Disconnect in progress</td>
</tr>
</tbody>
</table>

\texttt{extent\_addr}

When CSRC4QEX completes, the doubleword variable specified for \texttt{extent\_addr} contains the address of the extent.

\texttt{extent\_len}

When CSRC4QEX completes, the doubleword variable specified for \texttt{extent\_len} contains the length of the extent, in bytes.

\texttt{area\_addr}

When CSRC4QEX completes, the doubleword variable specified for \texttt{area\_addr} contains the address of cell storage.

\texttt{area\_size}

When CSRC4QEX completes, the doubleword variable specified for \texttt{area\_size} contains the size of cell storage for the extent.

\texttt{total\_cells}

When CSRC4QEX completes, the doubleword variable specified for \texttt{total\_cells} contains the total number of cells associated with the extents.

\texttt{avail\_cells}

When CSRC4QEX completes, the doubleword variable specified for \texttt{avail\_cells} contains the total number of cells associated with the specified extent that are available for allocation.

\texttt{return\_code}

When CSRC4QEX completes, the fullword variable specified for \texttt{return\_code} contains the return code.
CSRC4QEX callable service

ABEND codes

None.

Return and reason codes

When the CSRC4QEX service returns control to your program, GPR 15 (and return_code) contains one of the following return codes.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>Meaning: Program error. The extent number is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Make sure the extent number is within the range of 0 through 65536.</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>Meaning: Program error or system error. An extent chain was broken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>68</td>
<td>104</td>
<td>Meaning: Program error or system error. An extent chain is circular.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>6C</td>
<td>108</td>
<td>Meaning: Program error or system error. An extent could not be found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address for the right cell pool is being passed.</td>
</tr>
<tr>
<td>74</td>
<td>116</td>
<td>Meaning: Program error or system error. An extent has been overlaid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
Chapter 49. CSRC4QPL — Query the cell pool

Description

Call the CSRC4QPL cell pool service to receive status information about the cell pool.

CSRC4QPL does not prevent other programs from changing the pool during or after a query. CSRC4QPL returns the status as it was at the time you issued the CALL.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). All parameter areas, including the parameter list, may reside above 2GB.

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRC4QPL so the CALL macro can generate the correct code for AR mode.

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:

  * CSRC4_ANCHOR_LENGTH EQU 64

* Base length of the cell pool extent data area:

  * CSRC4_EXTENT_BASE EQU 192

* Length of the user-supplied pool name:

  * CSRC4_POOL_NAME_LEN EQU 8

*
CSRC4QPL callable service

Restrictions
None.

Input register information
Before calling the CSRC4QPL service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRC4QPL
,(cntl_alet
,anchor_addr
,user_name
,cell_size
,total_cells
,avail_cells
,number_extents
,return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:
CSRC4QPL callable service

\texttt{(cnt1\_alet)}

Specifies the fullword variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

\texttt{anchor\_addr}

Specifies the doubleword variable containing the address of the 64-byte anchor.

\texttt{user\_name}

When CSRC4QPL completes, the variable specified by \texttt{user\_name} contains the name on the CSRC4BLD service that created the cell pool.

\texttt{cell\_size}

When CSRC4QPL completes, the doubleword variable specified by \texttt{cell\_size} contains the size of each cell at the time the cell pool was created.

\texttt{total\_cells}

When CSRC4QPL completes, the doubleword variable specified by \texttt{total\_cells} contains the total number of cells associated with the extents.

\texttt{avail\_cells}

When CSRC4QPL completes, the doubleword variable specified by \texttt{avail\_cells} contains the total number of cells in active extents that are available for allocation.

\texttt{number\_extents}

When CSRC4QPL completes, the fullword variable specified by \texttt{number\_extents} contains the total number of extents (active or inactive, and connected or disconnected) in the cell pool.

\texttt{return\_code)}

When CSRC4QPL completes, the fullword variable specified by \texttt{return\_code} contains the return code.

**ABEND codes**

None.

**Return and reason codes**

When the CSRC4QPL service returns control to your program, GPR 15 (and \texttt{return\_code}) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>\textbf{Meaning:} The operation was successful. \textbf{Action:} None.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>\textbf{Meaning:} Program error. The anchor address is not valid. \textbf{Action:} Check to see if your program passed the wrong address or inadvertently overlaid the anchor area.</td>
</tr>
</tbody>
</table>
## CSRC4QPL callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 64                      | 100                 | **Meaning**: Program error or system error. The extent address is not valid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 50. CSRC4RFR — Return a cell to a cell pool (register interface)

Description
Call the CSRC4RFR cell pool service to return an allocated cell to a cell pool using the register interface, if your program cannot obtain storage for a parameter list. (The CSRC4RF1 service provides the same function with slightly enhanced performance.)

Environment
The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN.
- **AMODE**: 64-bit addressing mode.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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
As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area:**
  * CSRC4_ANCHOR_LENGTH EQU 64
- **Base length of the cell pool extent data area:**
  * CSRC4_EXTENT_BASE EQU 192
- **Length of the user-supplied pool name:**
  * CSRC4_POOL_NAME_LEN EQU 8

Restrictions
None.

Input register information
Before calling the CSRC4RFR service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:
CSRC4RFR callable service

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR 1</td>
<td>The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, CSRC4RFR ignores the value.</td>
</tr>
<tr>
<td>GPR 0</td>
<td>The doubleword address of the cell you want freed.</td>
</tr>
<tr>
<td>GPR 1</td>
<td>The doubleword anchor address.</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

CALL CSRC4RFR

Parameters

See “Input register information” on page 255.

ABEND codes

None.

Return and reason codes

When the CSRC4RFR service returns control to your program, GPR 15 contains one of the following return codes:
### CSRC4RFR callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning:** The operation was successful.  
**Action:** None. |
| 04                      | 04                  | **Meaning:** The last cell has been returned to an inactive extent.  
**Action:** None required. However, you might want to take some action depending on your application. |
| 1C                      | 28                  | **Meaning:** Program error. The anchor address is not valid.  
**Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | **Meaning:** Program error. The cell address is not valid.  
**Action:** Investigate the following possible causes:  
- The input cell address does not point to the beginning of a cell  
- The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | **Meaning:** Program error. Either you have already returned the cell or you never allocated it.  
**Action:** Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRC4RFR callable service
Chapter 51. CSRC4RF1 — Return a cell to a cell pool (register interface)

Description

Call the CSRC4RF1 cell pool service to return an allocated cell to a cell pool using the register interface, if your program cannot obtain storage for a parameter list. (The CSRC4RFR service provides the same function but slightly slower performance.)

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN.
- **AMODE:** 64-bit addressing mode.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
  * CSRC4_ANCHOR_LENGTH EQU 64
* Base length of the cell pool extent data area:
  * CSRC4_EXTENT_BASE EQU 192
* Length of the user-supplied pool name:
  * CSRC4_POOL_NAME_LEN EQU 8
* Length of the user-supplied savearea:
  * CSRC4_SAVEAREA_LEN EQU 216

Restrictions

None.
CSRC4RF1 callable service

Input register information
Before calling the CSRC4RF1 service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR 1</td>
<td>The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, CSRC4RF1 ignores the value.</td>
</tr>
<tr>
<td>GPR 0</td>
<td>The doubleword address of the cell you want freed.</td>
</tr>
<tr>
<td>GPR 1</td>
<td>The doubleword anchor address.</td>
</tr>
<tr>
<td>GPR 13</td>
<td>The address of a 216-byte save area that your program provides. The system does not change the first 8 bytes of this area.</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram.

```
CALL CSRC4RF1
```

Parameters
See "Input register information."

ABEND codes
None.
# CSRC4RF1 callable service

## Return and reason codes

When the CSRC4RF1 service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning**: The operation was successful.  
**Action**: None. |
| 04                      | 04                  | **Meaning**: The last cell has been returned to an inactive extent.  
**Action**: None required. However, you might want to take some action depending on your application. |
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
**Action**: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | **Meaning**: Program error. The cell address is not valid.  
**Action**: Investigate the following possible causes:  
- The input cell address does not point to the beginning of a cell  
- The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | **Meaning**: Program error. Either you have already returned the cell or you never allocated it.  
**Action**: Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning**: Program error or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRC4RF1 callable service
Chapter 52. CSRC4RGT — Allocate a cell from a cell pool (register interface)

Description

Call the CSRC4RGT cell pool service to allocate a cell from the cell pool using the register interface, if your program cannot obtain storage for a parameter list. CSRC4RGT allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to highest cell address. (The CSRC4RG1 service provides the same function with slightly enhanced performance.)

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN.
- **AMODE:** 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  - CSRC4_ANCHOR_LENGTH EQU 64
  - Base length of the cell pool extent data area:
  - CSRC4_EXTENT_BASE EQU 192
  - Length of the user-supplied pool name:
  - CSRC4_POOL_NAME_LEN EQU 8

Restrictions

None.
CSRC4RGT callable service

Input register information

Before calling the CSRC4RGT service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR 1</td>
<td>The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, CSRC4RGT ignores the value.</td>
</tr>
<tr>
<td>GPR 1</td>
<td>The doubleword anchor address</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

CALL CSRC4RGT

Parameters

See “Input register information.”

ABEND codes

None.

Return and reason codes

When the CSRC4RGT service returns control to your program, GPR 15 contains one of the following return codes:
<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                 | **Meaning:** The operation was successful.  
**Action:** None. |
| 08                      | 08                 | **Meaning:** Program error. There were no available cells in the pool.  
**Action:** Retry the request one or more times. If the problem persists, consider freeing existing cells, adding new cells to the cell pool, or both. |
| 1C                      | 28                 | **Meaning:** Program error. The anchor address is not valid.  
**Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 64                      | 100                | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRC4RGT callable service
Chapter 53. CSRC4RG1 — Allocate a cell from a cell pool (register interface)

Description

Call the CSRC4RG1 cell pool service to allocate a cell from the cell pool using the register interface, if your program cannot obtain storage for a parameter list. CSRC4RG1 allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to highest cell address. (The CSRC4RGT service provides the same function but slightly slower performance.)

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN.
AMODE: 64-bit addressing mode. All input addresses must be valid 64-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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

As the program must be running in AMODE 64 to call this service, be sure to issue SYSSTATE AMODE64=YES at the point(s) where the program begins running in AMODE 64.

Before you use cell pool services, you can optionally include the CSRC4ASM macro to generate cell pool services equate (EQU) statements. CSRC4ASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
  * CSRC4_ANCHOR_LENGTH EQU 64

* Base length of the cell pool extent data area:
  * CSRC4_EXTENT_BASE EQU 192

* Length of the user-supplied pool name:
  * CSRC4_POOL_NAME_LEN EQU 8

* Length of the user-supplied savearea:
  * CSRC4_SAVEAREA_LEN EQU 216

Restrictions

None.
CSRC4RG1 callable service

Input register information

Before calling the CSRC4RG1 service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR 1</td>
<td>The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, CSRC4RG1 ignores the value.</td>
</tr>
<tr>
<td>GPR 1</td>
<td>The doubleword anchor address</td>
</tr>
<tr>
<td>GPR 13</td>
<td>The address of a 216-byte save area that your program provides. The system does not change the first 8 bytes of this area.</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code in low 32 bits. High 32 bits are used as a work area by the system.</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

CALL CSRC4RG1

Parameters

See "Input register information."

ABEND codes

None.
Return and reason codes

When the CSRC4RG1 service returns control to your program, GPR 15 contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning**: The operation was successful.  
**Action**: None. |
| 08                      | 08                  | **Meaning**: Program error. There were no available cells in the pool.  
**Action**: Retry the request one or more times. If the problem persists, consider freeing existing cells, adding new cells to the cell pool, or both. |
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
**Action**: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning**: Program error or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 54. CSREVV — View an object and sequentially access it

Description

Call the CSREVV window service if your program references data in a sequential manner and you want to:

- Map a window to one or more blocks (4096 bytes) of a data object. If you specified scrolling when you called CSRIDAC to identify the object, CSREVV maps the window to the blocks in the scroll area and maps the scroll area to the object.
- Specify how many blocks window services is to try to transfer into the window each time CSREVV needs more data from the object.

Mapping a data object enables your program to access the data that is viewed through the window the same way it accesses other data in your storage.

The CSREVV and CSRVIEW services differ on how to specify sequential access:

- If you use CSRVIEW and specify sequential, when you reference data that is not in your window, window services reads up to 16 blocks — the one that contains the data your program requests, plus the next 15 consecutive blocks. The number of blocks that actually come into the window depends on the size of the window and the availability of central storage.
- If you use CSREVV, you can specify the number of additional consecutive blocks that window services reads into the window at one time. The number ranges from 0 through 255 blocks. The number of blocks that actually come into the window depends on the size of the window and the availability of central storage.

Use CSREVV if your program can benefit from having more than 16 blocks come into a window at one time, or fewer than 16 blocks at one time.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task
Cross memory mode: PASN = HASN = SASN
AMODE: 24- or 31-bit, but all addresses must be 31-bit
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Must be in the primary address space

Programming requirements

None.

Restrictions

The caller must follow all the restrictions imposed by the DIV macro.
CSREVW callable service

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a standard 18-word save area</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSREVW
\(\text{(operation\_type,object\_id,offset,span,window\_name,usage,disposition,pfcount,return\_code,reason\_code)}\)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

\(\text{(operation\_type)}\)
Specifies that you are to begin viewing an object.
Define **operation_type** as character data of length at least 5 bytes, containing the characters “BEGIN”.

**object_id**
Specifies the object identifier. Supply the object identifier that CSRIDAC returned when you obtained access to the object.

Define **object_id** as character data of length 8.

**offset**
Specifies the offset of the view into the object. Specify the offset in blocks of 4096 bytes.

Define **offset** as integer data of length 4.

**span**
Specifies the window size in blocks of 4096 bytes.

Define **span** as integer data of length 4.

**window_name**
Specifies the symbolic name you assigned to the window in your address space.

**usage**
Specifies that the expected pattern of references to data in the object will be sequential.

Define **usage** as character data of at least 4 bytes, containing the characters “SEQ”. Pad the string on the right with 1 or more blanks.

**disposition**
Defines how CSREVW is to handle data that is in the window when you begin a view. You can specify CSREVW BEGIN with a disposition of REPLACE or RETAIN. REPLACE and RETAIN cause the data in the window to be handled as follows:

**REPLACE**
The first time you reference a block to which the window is mapped, CSREVW replaces the data in the window with the data from the referenced block.

**RETAIN**
When you reference a block to which the window is mapped, the data in the window remains unchanged. When you call CSRSAVE to save the mapped blocks, CSRSAVE saves all of the mapped blocks because CSRSAVE considers them changed.

Define **disposition** as character data of length 7. If you specify RETAIN, pad the string on the right with 1 blank.

**pfcount**
Specifies the number of additional blocks you want window services to bring into the window each time your program references data that is not already in the window. The number you specify is added to the minimum of one block that window services always brings in. That is, if you specify a value of 20, window services brings in up to 21. The number of additional blocks ranges from zero through 255.

Define **pfcount** as integer data of length 4.

**return_code**
When CSREVW completes, **return_code** contains the return code. Define **return_code** as integer data of length 4.
CSREVW callable service

When CSREVW completes, reason_code contains the reason code. Define reason_code as integer data of length 4.

ABEND codes

CSREVW might abnormally terminate with abend code X'019'. See z/OS MVS System Codes for an explanation and programmer responses.

Return and reason codes

When the CSREVW service returns control to your program, GPR 15 (and return_code) contains a return code. GPR 0 (and reason_code) contains a reason code. The following table identifies return code and reason code combinations and explains their meanings. Data-in-virtual reason codes, which are returned with CSREVW return codes X'4' and X'C', are two bytes long and right justified. They are explained in the description of the DIV macro [Chapter 88, “DIV — Data-in-virtual,” on page 463].

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>00000004</td>
<td>00000165</td>
<td>Meaning: System error. The service could not retain all the data that was in the scroll area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Retry the request. If the problem persists, contact the appropriate IBM support personnel.</td>
</tr>
<tr>
<td>00000004</td>
<td>xxxxn000</td>
<td>Meaning: The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: See the DIV macro description for an explanation of reason code nnnn.</td>
</tr>
<tr>
<td>0000000C</td>
<td>xxxxn000</td>
<td>Meaning: The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: See the DIV macro description for an explanation of reason code nnnn.</td>
</tr>
<tr>
<td>0000002C</td>
<td>00000004</td>
<td>Meaning: Program error. Window services have not been defined to your system, or the link to the service failed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: If window services are available on your system, rerun the program one or more times. If the problem persists, contact the appropriate IBM support personnel.</td>
</tr>
</tbody>
</table>
Chapter 55. CSRIDAC — Request or terminate access to a data object

Description

Use the CSRIDAC callable window service to control access to a data object. The CSRIDAC service allows you to:

- Request access to a data object
- End access to a data object.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task
- **Cross memory mode**: PASN=HASN=SASN
- **AMODE**: 24- or 31-bit, but all addresses must be 31-bit addresses
- **ASC mode**: Primary
- **Interrupt status**: Enabled for I/O and external interrupts
- **Locks**: No locks held
- **Control parameters**: Must be in the primary address space

Programming requirements

None.

Restrictions

The caller must follow all the restrictions imposed by the DIV macro.

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a standard 18-word save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

None.

Syntax

Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRIDAC
   , (operation_type
   , object_type
   , object_name
   , scroll_area
   , object_state
   , access_mode
   , object_size
   , object_id
   , high_offset
   , return_code
   , reason_code)
```

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

(operation_type)

Specifies the type of operation the service is to perform:

- To request access to an object, specify BEGIN.
- To terminate access to an object, specify END. If the object is temporary, CSRIDAC deletes it.

Define operation_type as character data of length 5. If you specify END, pad the string on the right with blanks.

(object_type)

Specifies the type of object. The types are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DDNAME</td>
<td>The object is an existing (OLD) VSAM linear data set allocated to the file whose DDNAME is specified by object_name.</td>
</tr>
<tr>
<td>DSNAME</td>
<td>The object is the linear VSAM data set whose name is specified by object_name. The data set may already exist or may be a new data set that you want window services to create.</td>
</tr>
<tr>
<td>TEMPSPACE</td>
<td>The object is a temporary data object. Window services will delete the object when your program issues CSRIDAC END.</td>
</tr>
</tbody>
</table>

If operation_type is BEGIN, you must supply a value.

Define this parameter as character data of length 9. If you specify either DDNAME or DSNAME, pad the string on the right with blanks.
CSRIDAC callable service

.object_name
   Specifies the data set name of a permanent object or the DDNAME of a data
definition (DD) statement that defines a permanent object.
   • If object_type is DDNAME, object_name must contain the name of a DD
     statement.
   • If object_type is DSNAME, object_name must contain the data set name of
     the permanent object.

If operation_type is BEGIN and object_type is DDNAME or DSNAME, you must
supply a value for object_name.

Define object_name as character data of length 1 to 44. If object_name
contains fewer than 44 characters, pad the name on the right with blanks.

.scroll_area
   Specifies whether window services is to create a scroll area for the data object.
   YES      Create a scroll area.
   NO       Do not create a scroll area.

If operation_type is BEGIN and object_type is TEMPSPACE, specify YES.

Define scroll_area as character data of length 3. If you specify NO, pad the
string on the right with a blank.

.object_state
   Specifies the state of the object.
   OLD      The object exists.
   NEW      The object does not exist and window services must create it.

If operation_type is BEGIN and object_type is DSNAME, you must supply a
value for object_state.

Define object_state as character data of length 3.

.access_mode
   Specifies the type of access required.
   READ     READ access.
   UPDATE   UPDATE access.

If operation_type is BEGIN and object_type is DDNAME or DSNAME, you must
supply a value for access_mode. For a new or temporary data object, window
services assumes UPDATE.

Define access_mode as character data of length 6. If you specify READ, pad
the string on the right with 1 or 2 blanks.

.object_size
   Specifies the maximum size of the new object in units of 4096 bytes.

This parameter is required if either of the following conditions is true:
   • Operation_type is BEGIN, object_type is DSNAME, and object_state is NEW
   • Operation_type is BEGIN and object_type is TEMPSPACE

Define object_size as integer data of length 4.

.object_id
   Specifies the object identifier.

When operation_type is BEGIN, the service returns the object identifier in this
parameter. Use the identifier to identify the object to other window services.

When operation_type is END, you must supply the object identifier in this
parameter.
CSRIDAC callable service

Define object_id as character data of length 8.

,high_offset
When CSRIDAC completes, high_offset contains the size of the existing object expressed in blocks of 4096 bytes.

Define high_offset as integer data of length 4.

$return_code$
When CSRIDAC completes, return_code contains the return code. Define return_code as integer data of length 4.

(reason_code)
When CSRIDAC completes, reason_code contains the reason code. Define reason_code as integer data of length 4.

ABEND codes

The CSRIDAC service might abnormally terminate with abend code X'019'. See z/OS MVS System Codes for an explanation and programmer responses.

Return and reason codes

When the CSRIDAC service returns control to your program, return_code contains a return code and reason_code contains a reason code. The following table identifies return code and reason code combinations and explains their meanings.

Data-in-virtual reason codes, which are returned with CSRIDAC return codes X'4' and X'C', are two bytes long and right justified. They are explained in the description of the DIV macro (Chapter 88, “DIV — Data-in-virtual,” on page 463).

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000</td>
<td><strong>Meaning:</strong> The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>00000004</td>
<td>xxxxxxxxxxxx</td>
<td><strong>Meaning:</strong> Program error or environmental error. The operation was successful; however, data-in-virtual issued a warning. The value $n$ is a data-in-virtual reason code. The value $xxxx$ is not part of the intended programming interface.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> See the description of the DIV macro for an explanation of reason code $n$.</td>
</tr>
<tr>
<td>00000008</td>
<td>00000118</td>
<td><strong>Meaning:</strong> Environmental error. The system could not obtain enough storage to create a hiperspace for the temporary object or the scroll area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Rerun the program one or more times. If the problem persists, notify your system programmer, who can increase the SMF limit. The SMF limit, which is set by the installation, restricts the amount of virtual storage that programs in each address space can use for data spaces and hiperspaces.</td>
</tr>
</tbody>
</table>
### CSRIDAC callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000008                | 00000119                | Meaning: Environmental error. The system could not delete or unidentify the temporary object or the scroll area.  
Action: Retry the request. If the problem persists, record the return and reason code, and contact the appropriate IBM support personnel. |
| 00000008                | 0000011A                | Meaning: Environmental error. The system was unable to create a new VSAM linear data set. Your system must include SMS, and SMS must be active.  
Action: Contact your system programmer to request that SMS be made active. |
| 0000000C                | xxxxnnnn                | Meaning: Program error or environmental error. The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.  
Action: See the description of the DIV macro for an explanation of reason code nnnn. |
| 00000010                | rrrrnnnn                | Meaning: Program or environmental error. The system was unable to allocate or unallocate the data set specified as object_name. The value rrr is the return code from dynamic allocation. The value nnn is the two-byte reason code from dynamic allocation.  
Action: If object_state is new, make sure that a data set of the same name does not already exist. If this is the case, either use the existing data set or change the name of your data set. If you are unable to correct the problem, notify your system programmer. |
| 0000002C                | 00000004                | Meaning: Program error. Window services have not been defined to your system, or the link to the service failed.  
Action: If window services are available on your system, rerun the program one or more times. If the problem persists, contact the appropriate IBM support personnel. |
CSRIDAC callable service
Chapter 56. CSRL16J — Transfer control with all registers intact

Description

Call the CSRL16J service to transfer control to another routine running under the same request block (RB). The CSRL16J service functions much like a branch instruction, but will transfer control with the contents of all registers intact.

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

- Before calling the CSRL16J service, you must build data area CSRYL16J to form a parameter list that defines the entry characteristics and register contents for the target routine; include the CSRYL16J mapping macro. See [z/OS MVSP0000 Programming: Assembler Services Guide](http://www.ibm.com/support/docview.wss?uid=swg27000000) for information on how to build the parameter list.
- You can optionally include the CSRLJASM macro to obtain assembler declarations in the calling program for the return code from CSRL16J. CSRLJASM provides the following constants for use in your program:

```assembly

'**********************************************
* Service Return Codes                        *
'**********************************************
CSRL16J_OK EQU 0
CSRL16J_BAD_VERSION EQU 4
CSRL16J_BAD_AMODE EQU 8
CSRL16J_BAD_RESERVED EQU 12
CSRL16J_BAD_LENGTH EQU 16
CSRL16J_BAD_PSW EQU 24

'**********************************************
```

Restrictions

The caller cannot have an EUT FRR established.

Input register information

Before calling the CSRL16J service, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of a standard 18 word save area</td>
</tr>
</tbody>
</table>
CSRL16J callable service

Output register information

The CSRL16J service returns control to the caller only when it cannot successfully transfer control to the target routine because of an error. Otherwise CSRL16J transfers control to the target routine, which can return control to any program running under the same RB, including the calling program.

When CSRL16J returns control to the caller because of an error, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

When CSRL16J returns control to the caller because of an error, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRL16J,(L16J,return_code)
```

Parameters

The parameters are explained as follows:

L16J

Specifies the parameter list (CSRYL16J) containing the entry characteristics and environment for the target routine.

return_code

Specifies a fullword to contain the return code from the CSRL16J service.

ABEND codes

None.

Return and reason codes

If the CSRL16J service returns control to the caller, CSRL16J was unable to transfer control to the target routine. In this case, return_code contains a nonzero value.
When the CSRL16J service successfully transfers control to the target routine, *return_code* contains a value of zero.

Return codes from the CSRL16J service are as follows:

*Table 10. Return Codes for the CSRL16J Service*

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td><strong>Meaning</strong>: Successful completion. The calling program does not receive this return code because it indicates that the target routine received control.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: None.</td>
</tr>
<tr>
<td>04</td>
<td><strong>Meaning</strong>: The value specified in the L16JVERSION field of the L16J data area was not zero. The L16JVERSION field must contain a value of zero.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: When you build CSRYL16J, first clear the entire data area and then fill in the required fields. This process ensures that all fields that must contain zeros are correct.</td>
</tr>
<tr>
<td>08</td>
<td><strong>Meaning</strong>: The calling program was not in 31-bit addressing mode, which is required.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Make sure the calling program is in 31-bit addressing mode.</td>
</tr>
<tr>
<td>0C</td>
<td><strong>Meaning</strong>: One of the fields in CSRYL16J that is reserved for IBM use contained a nonzero value. Any field reserved for IBM use must contain a value of zero.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: When you build CSRYL16J, first clear the entire data area and then fill in the required fields. This process ensures that all fields that must contain zeros are correct.</td>
</tr>
<tr>
<td>10</td>
<td><strong>Meaning</strong>: The value specified in field L16JLENGTH in CSRYL16J was less than the actual length of the data area.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Make sure that the value in the L16JLENGTH field reflects the actual length of the data area.</td>
</tr>
<tr>
<td>18</td>
<td><strong>Meaning</strong>: The PSW provided in field L16JPSW of CSRYL16J specified an ASC mode that is not valid.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: In the L16JPSW field, specify either primary or AR ASC mode.</td>
</tr>
</tbody>
</table>
CSRL16J callable service
Chapter 57. CSRPACT — Activate previously connected storage

Description
Call the CSRPACT cell pool service to activate the extent cell storage for allocation. You must specify which extent you want to activate.

Environment
The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements
If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPACT so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPSAM macro to generate cell pool services equate (EQU) statements. CSRCPSAM provides the following constants for use in your program:

- **Length of the cell pool anchor data area**:
  - CSR_ANCHOR_LENGTH EQU 64
- **Base length of the cell pool extent data area**:
  - CSR_EXTENT_BASE EQU 128
- **Length of the user-supplied pool name**:
  - CSR_POOL_NAME_LEN EQU 8

Restrictions
None.
CSRPACT callable service

Input register information
Before calling the CSRPACT service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```assembly
CALL CSRPACT
   , (cntl_alet
   , anchor_addr
   , extent_num
   , return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- `,(cntl_alet
  Specifies the variable containing the ALET identifying the location of the anchor and extents. Initialize the ALET to 0 if your program is in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

- `,anchor_addr
  Identifies the variable containing the address of the 64-byte anchor.
Identifies the variable containing the number of the extent to be connected. The extent number must be within the range 0 to 65536.

When CSRPACT completes, the variable specified by return_code contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRPACT service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | Meaning: The operation was successful.  
Action: None. |
| 1C                      | 28                  | Meaning: Program error. The anchor address is not valid.  
Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 30                      | 48                  | Meaning: Program error. The extent number is not valid.  
Action: Make sure the extent number is within the range 0 to 65536. |
| 34                      | 52                  | Meaning: Program error. The extent is in the incorrect state.  
Action: Check to see if your program passed the wrong extent number. Make sure the extent is not already in an active state (that is, it has not been activated through CSRPACT or CSRPEXP). |
| 64                      | 100                 | Meaning: Program or system error. An extent chain was broken.  
Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | Meaning: Program or system error. An extent chain is circular.  
Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 6C                      | 108                 | Meaning: Program or system error. An extent could not be found.  
Action: Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
CSRPACT callable service
Chapter 58. CSRPBLD — Build a cell pool and initialize an anchor

Description

Call the CSRPBLD cell pool service to format a 64-byte area for the cell pool anchor. You must first have acquired the storage for the anchor. You can call this service only once for a given cell pool.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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: All parameters must reside in a single address or data space, and must be addressable by the caller. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call the CSRPBLD service so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
  * CSR_ANCHOR_LENGTH EQU 64
  *
  * Base length of the cell pool extent data area:
    * CSR_EXTENT_BASE EQU 128
    *
    * Length of the user-supplied pool name:
      * CSR_POOL_NAME_LEN EQU 8

Restrictions

None.
CSRPBLD callable service

Input register information
Before calling the CSRPBLD service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPBLD
    , (cnt1_alet
    , anchor_addr
    , user_name
    , cell_size
    , return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

,,(cnt1_alet
   Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

,,anchor_addr
   Specifies the variable containing the address of the cell pool anchor.
CSRPBLD callable service

 `,user_name`
  Specifies the 8-byte variable containing the name you want the service to assign to the pool. There are no restrictions on the name.

 `,cell_size`
  Specifies the variable containing the cell size in this pool. You can use any positive binary or hexadecimal number as the cell size.

 `,return_code`
  When CSRPBLD completes, `return_code` contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRPBLD service returns control to your program, GPR 15 (and `return_code`) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>18</td>
<td>24</td>
<td>Meaning: Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program passed the wrong anchor address. If the anchor is in a data space, make sure the anchor address is at least 63 bytes less than the address of the last byte of the data space.</td>
</tr>
<tr>
<td>44</td>
<td>68</td>
<td>Meaning: Program error. The cell size is not valid: it cannot be negative or 0.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Specify a positive value for the cell size.</td>
</tr>
</tbody>
</table>
CSRPBLD callable service
Chapter 59. CSRPCON — Connect cell storage to an extent

Description

Call the CSRPCON cell pool service to connect cell storage to the extent that you specify or to reuse a disconnected extent. The CSRPEXP service returned the extent number. The extent must be in the disconnected state, which means that you have not called CSRPACT to activate this particular extent.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in a primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPCON so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

\* Length of the cell pool anchor data area:

\*  

\* CSR_ANCHOR_LENGTH EQU 64
\*  

\* Base length of the cell pool extent data area:

\*  

\* CSR_EXTENT_BASE EQU 128
\*  

\* Length of the user-supplied pool name:

\*  

\* CSR_POOL_NAME_LEN EQU 8
\*  

Restrictions

None.
CSRCON callable service

Input register information
Before calling the CSRCON service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRCON
   , (cntl_alet
   , anchor_addr
   , area_addr
   , area_size
   , extent_num
   , return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
   , (cntl_alet
```

Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, you must code this parameter, even though any value that you code is ignored.
CSRCON callable service

anchor_addr
Specifies the variable containing the address of the 64-byte anchor.

area_addr
Specifies the variable containing the starting address of the cell storage area. The starting address of this area must be consistent with any cell boundary requirements that you might have.

area_size
Specifies the variable containing the length of the cell storage area. CSRCON determines the number of cells that will fit in the area.

extent_num
Specifies the variable containing the number of the extent to be connected. The extent number must be within the range 0 to 65536.

return_code
When CSRCON completes, the variable specified by return_code contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRCON service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning:** The operation was successful.  
|                         |                     | **Action:** None. |
| 1C                      | 28                  | **Meaning:** Program error. The anchor address is not valid.  
|                         |                     | **Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 30                      | 48                  | **Meaning:** Program error. The extent number is not valid.  
|                         |                     | **Action:** Specify the extent number within the range 0 to 65536. |
| 34                      | 52                  | **Meaning:** Program error. You issued the services in the wrong order, or did not issue a necessary service.  
|                         |                     | **Action:** Check to see if your program passed the wrong extent number. Make sure that the extent is in a disconnected state (that is, it has not been activated through CSRPRPACT or CSRPRPEXP). |
| 48                      | 72                  | **Meaning:** Program error. The cell area length is not valid.  
|                         |                     | **Action:** Check the specified cell area length. It should not be less than the cell size. |
CSRPCON callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4C                      | 76                  | **Meaning:** Program error. The service could not access the cell area address.  
**Action:** If the cell area is in a data space, make sure the cell area is completely within the data space. |
| 50                      | 80                  | **Meaning:** Program error. The cell area is too large.  
**Action:** Specify a larger extent size or a smaller cell area size. |
| 64                      | 100                 | **Meaning:** Program or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 6C                      | 108                 | **Meaning:** Program or system error. An extent could not be found.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
Chapter 60. CSRPDAC — Deactivate an extent

Description

Call the CSRPDAC cell pool service to deactivate a specific extent. Use this service to prepare the cell pool for contraction. You must specify which extent you want to deactivate.

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN.
- **AMODE:** 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPDAC so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  
  ```
  \* Length of the cell pool anchor data area:
  \*
  CSR_ANCHOR_LENGTH EQU 64
  \*
  ```

- Base length of the cell pool extent data area:
  
  ```
  \* Base length of the cell pool extent data area:
  \*
  CSR_EXTENT_BASE EQU 128
  ```

- Length of the user-supplied pool name:
  
  ```
  \* Length of the user-supplied pool name:
  \*
  CSR_POOL_NAME_LEN EQU 8
  ```

Restrictions

After calling CSRPDAC, you can still free (or return) cells, but you cannot get (or allocate) any others for this extent.
CSRPDAC callable service

Input register information
Before calling the CSRPDAC service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPDAC
    ,cntl_alet
    ,anchor_addr
    ,extent_num
    ,return_code
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

',cntl_alet
  Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

',anchor_addr
  Specifies the variable containing the address of the 64-byte anchor.
CSRPDAC callable service

,extent_num
   Specifies the variable containing the number of the extent that CSRPDAC will deactivate. The extent number must be within the range 0 to 65536.

,return_code
   When CSRPDAC completes, the variable specified for return_code contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRPDAC service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning:** The operation was successful.  
                          |                     | **Action:** None. |
| 04                      | 04                  | **Meaning:** The last cell has been returned to an inactive extent.  
                          |                     | **Action:** None required. However, you might take some action depending on your application. |
| 1C                      | 28                  | **Meaning:** Program error. The anchor address is not valid.  
                          |                     | **Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 30                      | 48                  | **Meaning:** Program error. The extent number is not valid.  
                          |                     | **Action:** Make sure the extent number is within the range 0 to 65536. |
| 34                      | 52                  | **Meaning:** Program error. You issued the services in the wrong order, or did not issue a necessary service.  
                          |                     | **Action:** Check to see if your program passed the wrong extent number. Make sure that the extent is in active state before calling the service. |
| 64                      | 100                 | **Meaning:** Program error or system error. An extent chain was broken.  
                          |                     | **Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
                          |                     | **Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
### CSRDPAC callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 6C                      | 108                 | **Meaning:** Program error or system error. An extent could not be found.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
Chapter 61. CSRPDIS — Disconnect the cell storage for an extent

Description

Call the CSRPDIS cell pool service to disconnect cell storage for a specific extent.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPDIS so the CALL macro can generate the correct code for AR mode.

Before you call CSRPDIS, you must have returned all cells associated with the extent and have called CSRPDAC to deactivate the extent.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area**:
  ```
  CSR_ANCHOR_LENGTH EQU 64
  *
  *
  * Base length of the cell pool extent data area:
  *
  CSR_EXTENT_BASE EQU 128
  *
  *
  * Length of the user-supplied pool name:
  *
  CSR_POOL_NAME_LEN EQU 8
  *
  *
  *
  *
  ```

Restrictions

None.
CSRPDIS callable service

Input register information
Before calling the CSRPDIS service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>4</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPDIS
   , (cntl_alet
   , anchor_addr
   , extent_num
   , area_addr
   , area_size
   , return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

`, (cntl_alet
  Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.
CSRPDIS callable service

anchor_addr
Specifies the variable containing the address of the 64-byte anchor.

extent_num
Specifies the variable containing the number of the extent that CSRPDIS will disconnect. The extent number must be within the range 0 to 65536.

area_addr
When CSRPDIS completes, the variable specified by area_addr contains the address of the disconnected storage area.

area_size
When CSRPDIS completes, the variable specified by area_size contains the size of the disconnected area.

return_code
When CSRPDIS completes, the variable specified by return_code contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRPDIS service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful. Action: None.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid. Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>30</td>
<td>48</td>
<td>Meaning: Program error. The extent number is not valid. Action: Make sure the extent number is within the range 0 to 65536.</td>
</tr>
<tr>
<td>34</td>
<td>52</td>
<td>Meaning: Program error. You issued the services in the wrong order, or did not issue a necessary service. Action: Call CSRPDAC to deactivate the extent before calling CSRPDIS to disconnect the cell storage for the extent.</td>
</tr>
<tr>
<td>38</td>
<td>56</td>
<td>Meaning: Program error. The service cannot disconnect the extent because some cells are still allocated. Action: Return all the cells associated with the extent before calling CSRPDIS to disconnect the cell storage for the extent.</td>
</tr>
</tbody>
</table>
### CSRPDIS callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 64                      | 100                 | **Meaning**: Program or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 6C                      | 108                 | **Meaning**: Program or system error. An extent could not be found.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address being passed is for the right cell pool. |
Chapter 62. CSRPEXP — Expand a cell pool

Description

Call the CSRPEXP cell pool service to:
• Add an extent to the cell pool
• Assign a number to the extent
• Optionally, establish a connection between the extent and cell storage
• Optionally, make the cell storage available for allocation.

Note: If you are reusing an extent, use CSRPCON and CSRPACT instead of CSRPEXP.

If you specify zero for the cell storage size, CSRPEXP will add an extent to the cell pool, but will keep it in a disconnected state. When you specify the extent size, allow 128 bytes plus one byte per eight cells of cell storage. CSRPEXP allocates cells contiguously, starting at the address you specify. If you specify zero for the area length, CSRPEXP ignores the area address.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPEXP so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPSASM macro to generate cell pool services equate (EQU) statements. CSRCPSASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
  * CSR_ANCHOR_LENGTH EQU 64
* Base length of the cell pool extent data area:
  * CSR_EXTENT_BASE EQU 128
  *
Restrictions

None.

Input register information

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

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```assembly
CALL CSRPEXP
  .(cntl_alet
  .anchor_addr
  .extent_addr
  .extent_size
  .area_addr
  .area_size
  .extent_num
  .return_code)
```
Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

\( (\text{cnt1\_alet}) \)

Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

\( (\text{anchor\_addr}) \)

Specifies the variable containing the address of the 64-byte anchor.

\( (\text{extent\_addr}) \)

Specifies the variable containing the address of the extent.

\( (\text{extent\_size}) \)

Specifies the variable containing the size of the extent.

\( (\text{area\_addr}) \)

Specifies the variable containing the starting address of the cell storage area. The starting address of this area must be consistent with any boundary requirements that you might have.

\( (\text{area\_size}) \)

Specifies the variable containing the length (binary or hexadecimal) of the storage area for the cells.

\( (\text{extent\_num}) \)

When CSRPEXP completes, the variable specifying \text{extent\_num} contains the number of the extent to be connected. You will use this number on subsequent CALLs.

\( (\text{return\_code}) \)

When CSRPEXP completes, the variable specifying \text{return\_code} contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRPEXP service returns control to your program, GPR 15 (and \text{return\_code}) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>0C</td>
<td>12</td>
<td>Meaning: Program error. There are too many extents in the cell pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program contains a logic error that caused the limit of 65536 extents per cell pool to be exceeded. If your program works as expected, consider using a larger cell pool.</td>
</tr>
</tbody>
</table>
## CSRPEXP callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
**Action**: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 28                      | 40                  | **Meaning**: Program error. The service could not use the extent address.  
**Action**: If the extent is in a data space, make sure the extent address is at least 128 bytes less than the address of the last byte of the data space. Also make sure the extent area does not overlap the anchor area. |
| 2C                      | 44                  | **Meaning**: Program error. The extent length is not valid.  
**Action**: Correct the extent length. It cannot be less than 129 bytes. |
| 48                      | 72                  | **Meaning**: Program error. The cell area length is not valid.  
**Action**: Correct the cell area length. The cell area size cannot be less than the cell size. |
| 4C                      | 76                  | **Meaning**: Program error. The service could not use the cell area address.  
**Action**: If the cell area is in a data space, make sure the cell area is completely within the data space. |
| 50                      | 80                  | **Meaning**: Program error. The cell area is too large.  
**Action**: Specify a larger extent size or a smaller cell area size. |
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 70                      | 112                 | **Meaning**: Program error or system error. An anchor has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid the anchor area. |
| 74                      | 116                 | **Meaning**: Program error or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRPEXP callable service
CSRPEXP callable service
Chapter 63. CSRPFRE — Return a cell to a cell pool

Description

Call the CSRPFRE cell pool service to return an allocated cell to a cell pool. You must specify the address of the cell that you want to return. (The CSRPFR1 and CSRPFR2 services provide the same function with slightly enhanced performance. CSRPFR2 is preferred over CSRPFR1 when using multiple extents.)

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you CALL CSRPFRE so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area:**
  - CSR_ANCHOR_LENGTH EQU 64

- **Base length of the cell pool extent data area:**
  - CSR_EXTENT_BASE EQU 128

- **Length of the user-supplied pool name:**
  - CSR_POOL_NAME_LEN EQU 8

Restrictions

None.
**CSRPFRE callable service**

**Input register information**
Before calling the CSRPFRE service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

**Performance implications**
None.

**Syntax**
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPFRE
    , (cntl_alet
    , anchor_addr
    , cell_addr
    , return_code)
```

**Parameters**
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- `ctl_alet`
  Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

- `anchor_addr`
  Specifies the variable containing the address of the 64-byte anchor.
CSRPFRE callable service

`cell_addr`
Specifies the variable containing the address of the cell that CSRPFRE is to free.

`return_code`
When CSRPFRE completes, the variable specified for `return_code` contains the return code.

### ABEND codes
None.

### Return and reason codes
When the CSRPFRE service returns control to your program, GPR 15 (and `return_code`) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning**: The operation was successful.  
Action: None. |
| 04                      | 04                  | **Meaning**: The last cell has been returned to an inactive extent.  
Action: None required. However, you might take some action depending on your application. |
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | **Meaning**: Program error. The cell address is not valid.  
Action: Investigate the following possible causes:  
- The input cell address does not point to the beginning of a cell.  
- The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | **Meaning**: Program error. Either you have already returned the cell or you never allocated it.  
Action: Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
### CSRPFRE callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell. |
Chapter 64. CSRPFR1 — Return a cell to a cell pool

Description

Call the CSRPFR1 cell pool service to return an allocated cell to a cell pool. You must specify the address of the cell that you want to return. (The CSRPFRE service provides the same function but slightly slower performance. CSRPFR2 is preferred over CSRPFR1 when using multiple extents.)

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you CALL CSRPFR1 so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  - **CSR_ANCHOR_LENGTH** EQU 64

- Base length of the cell pool extent data area:
  - **CSR_EXTENT_BASE** EQU 128

- Length of the user-supplied pool name:
  - **CSR_POOL_NAME_LEN** EQU 8

Restrictions

None.
Input register information
Before calling the CSRPFR1 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPFR1
   ,cntl_alet
   ,anchor_addr
   ,cell_addr
   ,return_code
   ,save_area)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
, (cnt1_alet
   Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

,anchor_addr
   Specifies the variable containing the address of the 64-byte anchor.
```
CSRPFR1 callable service

,cell_addr
   Specifies the variable containing the address of the cell that CSRPFR1 is to free.

$return_code$
   When CSRPFR1 completes, the variable specified for return_code contains the return code.

,save_area
   Specifies a 144-byte save area. The system does not change the first 8 bytes or the last 8 bytes of this area.

ABEND codes

None.

Return and reason codes

When the CSRPFR1 service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>04</td>
<td>04</td>
<td>Meaning: The last cell has been returned to an inactive extent.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None required. However, you might take some action depending on your application.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>54</td>
<td>84</td>
<td>Meaning: Program error. The cell address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Investigate the following possible causes:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The input cell address does not point to the beginning of a cell.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• The cell is not in the cell pool specified by the anchor address.</td>
</tr>
<tr>
<td>58</td>
<td>88</td>
<td>Meaning: Program error. Either you have already returned the cell or you never allocated it.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program contains a logic error that caused this situation to occur.</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>Meaning: Program error or system error. An extent chain was broken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
### CSRPF1 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning**: Program error or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell. |
Chapter 65. CSRPFR2 — Return a cell to a cell pool

Description

Call the CSRPFR2 cell pool service to return an allocated cell to a cell pool. You must specify the address of the cell that you want to return. (The CSRPFRE service provides the same function but slightly slower performance. CSRPFR2 is preferred over CSRPFR1 when using multiple extents, as CSRPFR2 has an additional input parameter for the address of the extent containing the cell to be freed.)

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you CALL CSRPFR2 so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  * `CSR_ANCHOR_LENGTH` EQU 64
  *
- Base length of the cell pool extent data area:
  * `CSR_EXTENT_BASE` EQU 128
  *
- Length of the user-supplied pool name:
  * `CSR_POOL_NAME_LEN` EQU 8
  *

Restrictions

None.
CSRPFR2 callable service

Input register information
Before calling the CSRPFR2 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPFR2
    , (cnt1_alet
    , anchor_addr
    , cell_addr
    , return_code
    , save_area)
    , extent_addr
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

`,(cnt1_alet`
Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.
CSRPFR2 callable service

(anchor_addr)
Specifies the variable containing the address of the 64-byte anchor.

cell_addr
Specifies the variable containing the address of the cell that CSRPFR2 is to free.

(return_code)
When CSRPFR2 completes, the variable specified for return_code contains the return code.

(save_area)
Specifies a 144-byte save area. The system does not change the first 8 bytes or the last 8 bytes of this area.

(extent_addr)
Specifies the variable containing the address of the extent that contains the cell that CSRPFR2 is to free. This address was returned with the cell address in the extent_addr parameter on a previous CSRPGT2 call.

ABEND codes
None.

Return and reason codes
When the CSRPFR2 service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning:** The operation was successful.  
**Action:** None. |
| 04                      | 04                  | **Meaning:** The last cell has been returned to an inactive extent.  
**Action:** None required. However, you might take some action depending on your application. |
| 1C                      | 28                  | **Meaning:** Program error. The anchor address is not valid.  
**Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | **Meaning:** Program error. The cell address is not valid.  
**Action:** Investigate the following possible causes:  
• The input cell address does not point to the beginning of a cell.  
• The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | **Meaning:** Program error. Either you have already returned the cell or you never allocated it.  
**Action:** Check to see if your program contains a logic error that caused this situation to occur. |
### CSRPF2 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning**: Program error or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell. |
Chapter 66. CSRPGET — Allocate a cell from a cell pool

Description

Call the CSRPGET cell pool service to allocate a cell from the cell pool. CSRPGET allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to the highest cell address. CSRPGET passes back to the calling program the address of the cell it allocated but does not clear the cell storage to binary zeros. (The CSRPGT1 and CSRPGT2 services provide the same function with slightly enhanced performance. CSRPGT2 is preferred over CSRPGT1 when using multiple extents.)

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.

ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPGET so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
* CSR_ANCHOR_LENGTH EQU 64

* Base length of the cell pool extent data area:
* CSR_EXTENT_BASE EQU 128

* Length of the user-supplied pool name:
* CSR_POOL_NAME_LEN EQU 8
CSRPGT callable service

Restrictions
None.

Input register information
Before calling the CSRPGT service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPGET,
    (cntl_alet,
    anchor_addr,
    cell_addr,
    return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
, (cntl_alet)
```

Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.
CSRPGET callable service

anchor_addr
Specifies the variable containing the address of the 64-byte anchor.

cell_addr
When CSRPGET completes, the variable specified by cell_addr contains the address of the cell that CSRPGET allocated.

return_code)
When CSRPGET completes, the variable specified by return_code contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRPGET service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful. Action: None.</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td>Meaning: Program error. There were no available cells in the pool. More than one program could be using the cell pool. Action: Retry the request one or more times. If the problem persists, consider freeing existing cells or adding new cells to the cell pool, or both.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td>Meaning: Program error. The anchor address is not valid. Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td>Meaning: Program or system error. An extent chain was broken. Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>68</td>
<td>104</td>
<td>Meaning: Program or system error. An extent chain is circular. Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>74</td>
<td>116</td>
<td>Meaning: Program or system error. An extent has been overlaid. Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
CSRPGET callable service
Chapter 67. CSRPGT1 — Allocate a cell from a cell pool

Description

Call the CSRPGT1 cell pool service to allocate a cell from the cell pool. CSRPGT1 allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to the highest cell address. CSRPGT1 passes back to the calling program the address of the cell it allocated but does not clear the cell storage to binary zeros. (The CSRPGET service provides the same function but slightly slower performance. CSRPGT2 is preferred over CSRPGT1 when using multiple extents.)

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state with PSW key 8-15
- **Dispatchable unit mode:** Task or SRB
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode:** Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPGT1 so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  - CSR_ANCHOR_LENGTH EQU 64

- Base length of the cell pool extent data area:
  - CSR_EXTENT_BASE EQU 128

- Length of the user-supplied pool name:
  - CSR_POOL_NAME_LEN EQU 8

© Copyright IBM Corp. 1988, 2012
CSRPGT1 callable service

Restrictions

None.

Input register information

Before calling the CSRPGT1 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPGT1
 , (cnt1_alet
  , anchor_addr
  , cell_addr
  , return_code
  , save_area)
```

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- `(cnt1_alet)`
  Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR
mode and the anchor and extents are in the primary address space. If your
program is running in primary ASC mode, the value is ignored, but you must
code the parameter anyway.

anchor_addr
Specifies the variable containing the address of the 64-byte anchor.

cell_addr
When CSRPGT1 completes, the variable specified by cell_addr contains the
address of the cell that CSRPGT1 allocated.

return_code
When CSRPGT1 completes, the variable specified by return_code contains the
return code.

save_area
Specifies a 144-byte save area. The system does not change the first 8 bytes
or the last 8 bytes of this area.

ABEND codes
None.

Return and reason codes
When the CSRPGT1 service returns control to your program, GPR 15 (and
return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td><strong>Meaning:</strong> The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td><strong>Meaning:</strong> Program error. There were no available</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cells in the pool. More than one program could be</td>
</tr>
<tr>
<td></td>
<td></td>
<td>using the cell pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Retry the request one or more times. If the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>problem persists, consider freeing existing cells or</td>
</tr>
<tr>
<td></td>
<td></td>
<td>adding new cells to the cell pool, or both.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td><strong>Meaning:</strong> Program error. The anchor address is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check to see if your program passed the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>wrong anchor address or inadvertently overlaid the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>anchor area.</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td><strong>Meaning:</strong> Program or system error. An extent chain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>was broken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check to see if your program inadvertently</td>
</tr>
<tr>
<td></td>
<td></td>
<td>overlaid an extent area. Make sure that no extent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>belongs to more than one cell pool.</td>
</tr>
<tr>
<td>68</td>
<td>104</td>
<td><strong>Meaning:</strong> Program or system error. An extent chain</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is circular.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check to see if your program inadvertently</td>
</tr>
<tr>
<td></td>
<td></td>
<td>overlaid an extent area. Make sure that no extent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
### CSRPGT1 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 74                      | 116                 | **Meaning**: Program or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 68. CSRPGT2 — Allocate a cell from a cell pool

Description

Call the CSRPGT2 cell pool service to allocate a cell from the cell pool. CSRPGT2 allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to the highest cell address. CSRPGT2 passes back to the calling program the address of the cell it allocated but does not clear the cell storage to binary zeros. (The CSRPGT1 service provides the same function but slightly slower performance. CSRPGT2 is preferred over CSRPGT1 when using multiple extents, as CSRPGT2 has an additional output parameter to return the address of the extent containing the obtained cell.)

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller’s dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPGT2 so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area**:
  ```
  CSR_ANCHOR_LENGTH      EQU       64
  *
  *  Base length of the cell pool extent data area:
  ```
  ```
  CSR_EXTENT_BASE         EQU       128
  *
  ```
  ```
  *  Length of the user-supplied pool name:
  ```
  CSR_POOL_NAME_LEN       EQU       8
  *  ```
CSRPGT2 callable service

Restrictions
None.

Input register information
Before calling the CSRPGT2 service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPGT2
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
,(cnt1_alet
,anchor_addr
,cell_addr
,return_code
,save_area)
,extent_addr
```

Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR
CSRPGT2 callable service

mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

,anchor_addr
Specifies the variable containing the address of the 64-byte anchor.

cell_addr
When CSRPGT2 completes, the variable specified by cell_addr contains the address of the cell that CSRPGT2 allocated.

,return_code
When CSRPGT2 completes, the variable specified by return_code contains the return code.

,save_area
Specifies a 144-byte save area. The system does not change the first 8 bytes or the last 8 bytes of this area.

,extent_addr
Specifies the variable that is to contain the address of the extent containing the obtained cell. This address should be provided in the extent_addr parameter when using CSRPFR2 to free the cell.

ABEND codes
None.

Return and reason codes
When the CSRPGT2 service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | Meaning: The operation was successful.  
                        |                     | Action: None. |
| 08                      | 08                  | Meaning: Program error. There were no available cells in the pool. More than one program could be using the cell pool.  
                        |                     | Action: Retry the request one or more times. If the problem persists, consider freeing existing cells or adding new cells to the cell pool, or both. |
| 1C                      | 28                  | Meaning: Program error. The anchor address is not valid.  
                        |                     | Action: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 64                      | 100                 | Meaning: Program or system error. An extent chain was broken.  
                        |                     | Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
### CSRPGT2 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 68                      | 104                 | **Meaning:** Program or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning:** Program or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 69. CSRPQCL — Query a cell

Description
Call the CSRPQCL cell pool service to receive status information about a specified cell in a cell pool. CSRPQCL reports whether the cell is free or allocated, and returns the number of the extent associated with the cell. CSRPQCL does not prevent other programs from changing the pool during or after a query. CSRPQCL returns the status as it was at the time you issued the CALL.

Environment
The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements
If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPQCL so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
* CSR_ANCHOR_LENGTH EQU 64
* * Base length of the cell pool extent data area:
* CSR_EXTENT_BASE EQU 128
* * Length of the user-supplied pool name:
* CSR_POOL_NAME_LEN EQU 8
* *

Restrictions
None.
**CSRPQCL callable service**

**Input register information**
Before calling the CSRPQCL service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

**Performance implications**
None.

**Syntax**
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRPQCL
   ,(cntl_alet
   ,anchor_addr
   ,cell_addr
   ,cell_avail
   ,extent_num
   ,return_code)
```

**Parameters**
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

```
,(cntl_alet
   Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.
```
Specifies the variable containing the address of the 64-byte anchor.

Specifies the variable containing the address of the cell the service will query.

When CSRPQCL completes, the variable specified for cell_avail contains one of the following values. These indicate the status of the specified cell at the time you issued the CALL.

- 0: Cell available
- 1: Cell allocated

When CSRPQCL completes, the variable specified for extent_num contains the number of the extent that contains the specified cell.

When CSRPQCL completes, the variable specified for return_code contains the return code.

### ABEND codes

None.

### Return and reason codes

When the CSRPQCL service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning**: The operation was successful.  
                         |                     | **Action**: None. |
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
                         |                     | **Action**: Check to see if the program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | **Meaning**: Program error. The cell address is not valid.  
                         |                     | **Action**: Investigate the following possible causes:  
                         |                     | - The input cell address does not point to the beginning of a cell  
                         |                     | - The cell is not in the cell pool specified by the anchor address. |
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
                         |                     | **Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
### CSRPQCL callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 70. CSRPQEX — Query a cell pool extent

Description

Call the CSRPQEX cell pool service to receive status information about a specified extent.

CSRPQEX does not prevent other programs from changing the pool during or after a query. CSRPQEX returns the status as it was at the time you issued the CALL.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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 a single address or data space. Control parameters must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPQEX so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
* CSR_ANCHOR_LENGTH EQU 64
* * Base length of the cell pool extent data area:
* CSR_EXTENT_BASE EQU 128
* * Length of the user-supplied pool name:
* CSR_POOL_NAME_LEN EQU 8
* *

Restrictions

None.
CSRPGEX callable service

Input register information
Before calling the CSRPGEX service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
,(cntl_alet,
,anchor_addr
,extent_num
,status
,extent_addr
,extent_len
,area_addr
,area_size
,total_cells
,avail_cells
,return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

\(\text{,}(\text{cntl_alet})\)

Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR
CSRPQEX callable service

mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.

anchor_addr
Specifies the variable containing the address of the 64-byte anchor.

extent_num
Specifies the variable containing the number of the extent the service will query.

status
When CSRPQEX completes, the variable specified for status contains one of the following decimal numbers. These indicate the status of the extent at the time of the CALL.

- 1: Disconnected and inactive
- 2: Connect in progress
- 3: Connected and inactive
- 4: Connected and active
- 5: Disconnect in progress

extent_addr
When CSRPQEX completes, the variable specified for extent_addr contains the address of the extent.

extent_len
When CSRPQEX completes, the variable specified for extent_len contains the length of the extent, in bytes.

area_addr
When CSRPQEX completes, the variable specified for area_addr contains the address of cell storage.

area_size
When CSRPQEX completes, the variable specified for area_size contains the size of cell storage for the extent.

total_cells
When CSRPQEX completes, the variable specified for total_cells contains the total number of cells associated with the extents.

avail_cells
When CSRPQEX completes, the variable specified for avail_cells contains the total number of cells associated with the specified extent that are available for allocation.

return_code
When CSRPQEX completes, the variable specified for return_code contains the return code.

ABEND codes
None.

Return and reason codes
When the CSRPQEX service returns control to your program, GPR 15 (and return_code) contains one of the following return codes.
**CSRQPQEX callable service**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning:** The operation was successful.  
**Action:** None. |
| 1C                      | 28                  | **Meaning:** Program error. The anchor address is not valid.  
**Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 30                      | 48                  | **Meaning:** Program error. The extent number is not valid.  
**Action:** Make sure the extent number is within the range of 0 through 65536. |
| 64                      | 100                 | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 6C                      | 108                 | **Meaning:** Program error or system error. An extent could not be found.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that the anchor address for the right cell pool is being passed. |
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
Chapter 71. CSRPQPL — Query the cell pool

Description

Call the CSRPQPL cell pool service to receive status information about the cell pool.

CSRPQPL does not prevent other programs from changing the pool during or after a query. CSRPQPL returns the status as it was at the time you issued the CALL.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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 a single address or data space. They must be in the primary address space or in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If your program is in AR mode, issue the SYSSTATE macro with ASCENV=AR before you call CSRPQPL so the CALL macro can generate the correct code for AR mode.

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  - \texttt{CSR\_ANCHOR\_LENGTH} EQU 64
- Base length of the cell pool extent data area:
  - \texttt{CSR\_EXTENT\_BASE} EQU 128
- Length of the user-supplied pool name:
  - \texttt{CSR\_POOL\_NAME\_LEN} EQU 8

Restrictions

None.
CSRPQPL callable service

Input register information
Before calling the CSRPQPL service, the caller does not have to place any information into any register unless using it in register notation for a particular parameter, or using it as a base register.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

Syntax
Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
,(cnt1_alet,
,anchor_addr,
,user_name,
,cell_size,
,total_cells,
,avail_cells,
,number_extents,
,return_code)
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

`(cnt1_alet`
Specifies the variable containing the ALET that identifies the location of the anchor and extents. Initialize the ALET to 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, the value is ignored, but you must code the parameter anyway.
CSRPQPL callable service

(anchor_addr

Specifies the variable containing the address of the 64-byte anchor.

(user_name

When CSRPQPL completes, the variable specified by user_name contains the name on the CSRPBLD service that created the cell pool.

cell_size

When CSRPQPL completes, the variable specified by cell_size contains the size of each cell at the time the cell pool was created.

total_cells

When CSRPQPL completes, the variable specified by total_cells contains the total number of cells associated with the extents.

_avail_cells

When CSRPQPL completes, the variable specified by avail_cells contains the total number of cells in active extents that are available for allocation.

number_extents

When CSRPQPL completes, the variable specified by number_extents contains the total number of extents (active or inactive, and connected or disconnected) in the cell pool.

(return_code)

When CSRPQPL completes, the variable specified by return_code contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRPQPL service returns control to your program, GPR 15 (and return_code) contains one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | Meaning: The operation was successful.  
  Action: None.  |
| 1C                      | 28                  | Meaning: Program error. The anchor address is not valid.  
  Action: Check to see if your program passed the wrong address or inadvertently overlaid the anchor area.  |
| 64                      | 100                 | Meaning: Program error or system error. The extent address is not valid.  
  Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.  |
| 68                      | 104                 | Meaning: Program error or system error. An extent chain is circular.  
  Action: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.  |
CSRPQPL callable service
Chapter 72. CSRPRFR — Return a cell to a cell pool (register interface)

Description

Call the CSRPRFR cell pool service to return an allocated cell to a cell pool using the register interface, if your program cannot obtain storage for a parameter list. (The CSRPRFR1 service provides the same function with slightly enhanced performance.)

Environment

The requirements for the caller are:
- Minimum authorization: Problem state with PSW key 8-15
- Dispatchable unit mode: Task or SRB
- Cross memory mode: Any PASN, any HASN, any SASN.
- AMODE: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service.
- ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- 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

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- Length of the cell pool anchor data area:
  - CSR_ANCHOR_LENGTH EQU 64
- Base length of the cell pool extent data area:
  - CSR_EXTENT_BASE EQU 128
- Length of the user-supplied pool name:
  - CSR_POOL_NAME_LEN EQU 8

Restrictions

None.

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR 1</td>
<td>The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents</td>
</tr>
</tbody>
</table>
CSRPRFR callable service

are in the primary address space. If your program is running in primary ASC mode, CSRPRFR ignores the value.

| GPR 0 | The address of the cell you want freed. |
| GPR 1 | The anchor address. |

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

CALL CSRPRFR

Parameters

See "Input register information" on page 347.

ABEND codes

None.

Return and reason codes

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
</tbody>
</table>

348  z/OS V1R13.0 MVS Assembler Services Reference ABE-HSP
### CSRPRFR callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04                      | 04                  | **Meaning:** The last cell has been returned to an inactive extent.  
**Action:** None required. However, you might want to take some action depending on your application. |
| 1C                      | 28                  | **Meaning:** Program error. The anchor address is not valid.  
**Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | **Meaning:** Program error. The cell address is not valid.  
**Action:** Investigate the following possible causes:  
- The input cell address does not point to the beginning of a cell  
- The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | **Meaning:** Program error. Either you have already returned the cell or you never allocated it.  
**Action:** Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRPRFR callable service
Chapter 73. CSRPRFR1 — Return a cell to a cell pool (register interface)

Description

Call the CSRPRFR1 cell pool service to return an allocated cell to a cell pool using the register interface, if your program cannot obtain storage for a parameter list. (The CSRPRFR service provides the same function but slightly slower performance.)

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN.
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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

Before you use cell pool services, you can optionally include the CSRCPASM macro to generate cell pool services equate (EQU) statements. CSRCPASM provides the following constants for use in your program:

- **Length of the cell pool anchor data area**:
  - CSR_ANCHOR_LENGTH EQU 64
- **Base length of the cell pool extent data area**:
  - CSR_EXTENT_BASE EQU 128
- **Length of the user-supplied pool name**:
  - CSR_POOL_NAME_LEN EQU 8

Restrictions

None.

Input register information

Before calling the CSRPRFR1 service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR 1</td>
<td>The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents</td>
</tr>
</tbody>
</table>
CSRPRFR1 callable service

The address of the cell you want freed.
GPR 0
The anchor address.
GPR 1
The address of a 144-byte save area that your program provides.
The system does not change the first 8 bytes or the last 8 bytes of
this area.
GPR 13

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

CALL CSRPRFR1

Parameters

See "Input register information" on page 351.

ABEND codes

None.

Return and reason codes

When the CSRPRFR1 service returns control to your program, GPR 15 contains
one of the following return codes:

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td>Meaning: The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Action: None.</td>
</tr>
<tr>
<td>Hexadecimal Return Code</td>
<td>Decimal Return Code</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------</td>
<td>--------------------</td>
</tr>
</tbody>
</table>
| 04                      | 04                  | **Meaning**: The last cell has been returned to an inactive extent.  
**Action**: None required. However, you might want to take some action depending on your application. |
| 1C                      | 28                  | **Meaning**: Program error. The anchor address is not valid.  
**Action**: Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 54                      | 84                  | **Meaning**: Program error. The cell address is not valid.  
**Action**: Investigate the following possible causes:  
• The input cell address does not point to the beginning of a cell  
• The cell is not in the cell pool specified by the anchor address. |
| 58                      | 88                  | **Meaning**: Program error. Either you have already returned the cell or you never allocated it.  
**Action**: Check to see if your program contains a logic error that caused this situation to occur. |
| 64                      | 100                 | **Meaning**: Program error or system error. An extent chain was broken.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning**: Program error or system error. An extent chain is circular.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning**: Program error or system error. An extent has been overlaid.  
**Action**: Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRPRFR1 callable service
Chapter 74. CSRPRGT — Allocate a cell from a cell pool (register interface)

Description
Call the CSRPRGT cell pool service to allocate a cell from the cell pool using the register interface, if your program cannot obtain storage for a parameter list. CSRPRGT allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to highest cell address. (The CSRPRGT1 service provides the same function with slightly enhanced performance.)

Environment
The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN.
- **AMODE**: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
- **ASC mode**: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
- **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
Before you use cell pool services, you can optionally include the CSRCPSASM macro to generate cell pool services equate (EQU) statements. CSRCPSASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
  * CSR_ANCHOR_LENGTH EQU 64

* Base length of the cell pool extent data area:
  * CSR_EXTENT_BASE EQU 128

* Length of the user-supplied pool name:
  * CSR_POOL_NAME_LEN EQU 8

Restrictions
None.

Input register information
Before calling the CSRPRGT service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
CSRPRGT callable service

AR 1  The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, CSRPRGT ignores the value.

GPR 1  The anchor address

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Address of the allocated cell</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

CALL CSRPRGT

Parameters

See "Input register information" on page 355.

ABEND codes

None.

Return and reason codes

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | 00                  | **Meaning**: The operation was successful.  
**Action**: None. |
<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08                      | 08                  | **Meaning:** Program error. There were no available cells in the pool.  
**Action:** Retry the request one or more times. If the problem persists, consider freeing existing cells, adding new cells to the cell pool, or both. |
| 1C                      | 28                  | **Meaning:** Program error. The anchor address is not valid.  
**Action:** Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area. |
| 64                      | 100                 | **Meaning:** Program error or system error. An extent chain was broken.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 68                      | 104                 | **Meaning:** Program error or system error. An extent chain is circular.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
| 74                      | 116                 | **Meaning:** Program error or system error. An extent has been overlaid.  
**Action:** Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool. |
CSRPRGT callable service
Chapter 75. CSRRPRGT1 — Allocate a cell from a cell pool (register interface)

Description

Call the CSRRPRGT1 cell pool service to allocate a cell from the cell pool using the register interface, if your program cannot obtain storage for a parameter list. CSRRPRGT1 allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to highest cell address. (The CSRRPRGT service provides the same function but slightly slower performance.)

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN.
AMODE: 24- or 31-bit addressing mode. Nucleus-resident code must be in 31-bit addressing mode when calling the service. All input addresses must be valid 31-bit addresses.
ASC mode: Primary or AR mode. (If the anchor and the extents are located in a data space, the caller must be in AR mode.)
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

Before you use cell pool services, you can optionally include the CSRCPSASM macro to generate cell pool services equate (EQU) statements. CSRCPSASM provides the following constants for use in your program:

* Length of the cell pool anchor data area:
* CSR_ANCHOR_LENGTH EQU 64
* * Base length of the cell pool extent data area:
* CSR_EXTENT_BASE EQU 128
* * Length of the user-supplied pool name:
* CSR_POOL_NAME_LEN EQU 8
* *

Restrictions

None.

Input register information

Before calling the CSRRPRGT1 service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
</table>
CSRPRGT1 callable service

AR 1  The ALET used to access all the cell storage areas. Specify 0 if your program is running in AR mode and the anchor and extents are in the primary address space. If your program is running in primary ASC mode, CSRPRGT1 ignores the value.

GPR 1  The anchor address
GPR 13 The address of a 144-byte save area that your program provides. The system does not change the first 8 bytes or the last 8 bytes of this area.

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Address of the allocated cell</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

CALL CSRPRGT1

Parameters

See "Input register information" on page 359.

ABEND codes

None.

Return and reason codes

When the CSRPRGT1 service returns control to your program, GPR 15 contains one of the following return codes:
### CSRPRGT1 callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Decimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>00</td>
<td><strong>Meaning:</strong> The operation was successful.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None.</td>
</tr>
<tr>
<td>08</td>
<td>08</td>
<td><strong>Meaning:</strong> Program error. There were no available cells in the pool.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Retry the request one or more times. If the problem persists, consider freeing existing cells, adding new cells to the cell pool, or both.</td>
</tr>
<tr>
<td>1C</td>
<td>28</td>
<td><strong>Meaning:</strong> Program error. The anchor address is not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check to see if your program passed the wrong anchor address or inadvertently overlaid the anchor area.</td>
</tr>
<tr>
<td>64</td>
<td>100</td>
<td><strong>Meaning:</strong> Program error or system error. An extent chain was broken.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>68</td>
<td>104</td>
<td><strong>Meaning:</strong> Program error or system error. An extent chain is circular.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
<tr>
<td>74</td>
<td>116</td>
<td><strong>Meaning:</strong> Program error or system error. An extent has been overlaid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Check to see if your program inadvertently overlaid an extent area. Make sure that no extent belongs to more than one cell pool.</td>
</tr>
</tbody>
</table>
CSRPRGT1 callable service
Chapter 76. CSRREFR — Refresh an object

Description
To refresh changed data that is in a window, a scroll area, or a temporary object, call the CSRREFR window service. CSRREFR refreshes changed data within specified blocks as follows:

• If the object is permanent, CSRREFR replaces specified changed blocks in windows or the scroll area with corresponding blocks from the object on DASD.
• For a temporary object, CSRREFR refreshes specified changed blocks in windows and the object by setting the blocks to binary zeros.

Environment
The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 24- or 31-bit, but all addresses must be 31-bit addresses
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Must be in the primary address space

Programming requirements
None.

Restrictions
The caller must follow all the restrictions imposed by the DIV macro.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a standard 18-word save area</td>
</tr>
</tbody>
</table>

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Contains the reason code.</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
CSRREFR callable 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.

Performance implications

None.

Syntax

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRREFR
   ,object_id
   ,offset
   ,span
   ,return_code
   ,reason_code)
```

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- `object_id`
  - Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.
  - Define `object_id` as character data of length 8.

- `offset`
  - Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.
  - Define `offset` as integer data of length 4.
  - `offset` and `span`, together, determine what part of the object window services refreshes. To refresh the entire object, specify 0 for `offset` and 0 for `span`.

- `span`
  - Specifies how many 4096-byte blocks CSRREFR is to refresh.
  - Define `span` as integer data of length 4.

- `return_code`
  - When CSRREFR completes, `return_code` contains the return code. Define `return_code` as integer data of length 4.

- `reason_code`
  - When CSRREFR completes, `reason_code` contains the reason code. Define `reason_code` as integer data of length 4.

ABEND codes

CSRREFR might abnormally terminate with abend code X'019D'. See z/OS MVS System Codes for an explanation and programmer responses.
Return and reason codes

When the CSRREFR service returns control to your program, GPR 15 (and return_code) contains a return code and GPR 0 (and reason_code) contains a reason code. The following table identifies return code and reason code combinations and explains their meanings.

The data-in-virtual reason code, which is returned with CSRREFR return code X'C', is two bytes long and right justified. It is explained in the description of the DIV macro [Chapter 88, “DIV — Data-in-virtual,” on page 463].

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000000                | 00000000                | **Meaning:** The operation was successful.  
**Action:** None. |
| 00000008                | 00000152                | **Meaning:** Program error. The system could not refresh all the temporary objects within the specified span.  
**Action:** Investigate the following possible causes:  
• The window to be refreshed contains an I/O DEFINEd block  
• The data space in which the window is located is deleted. |
| 0000000C                | xxxxnnnn                | **Meaning:** The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.  
**Action:** See the DIV macro description for an explanation of nnnn. |
| 0000002C                | 00000004                | **Meaning:** Program error. Window services have not been defined to your system, or the link to the service failed.  
**Action:** If window services are available on your system, rerun the program one or more times. If the problem persists, record the return and reason code, and contact the appropriate IBM support personnel. |
CSRREFR callable service
Chapter 77. CSRSAVE — Save changes made to a permanent object

Description

To update specified blocks of a permanent object with changes, call the CSRSAVE window service. The changes can be in blocks that are mapped to the scroll area, in blocks that are mapped to windows, or in a combination of these places.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state with PSW key 8-15
- **Dispatchable unit mode**: Task
- **Cross memory mode**: PASN=HASN=SASN
- **AMODE**: 24- or 31-bit, but all addresses must be 31-bit addresses
- **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

You cannot use CSRSAVE to save changes made to a temporary object. If you call CSRSAVE for a temporary object, CSRSAVE ignores the request and returns control to your program with a return code of 8. To save changes made to a temporary object, call CSRSACT.

The caller must follow all the restrictions imposed by the DIV macro.

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a standard 18-word save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
CSRSAVE callable service

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

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

Performance implications
None.

Syntax
Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRSAVE
,.object_id
,.offset
,.span
,.new_hi_offset
,.return_code
,.reason_code
```

Parameters
All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

,.object_id
   Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.
   Define object_id as character data of length 8.

,.offset
   Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.
   Define offset as integer data of length 4.
   offset and span, together, determine what part of the object window services saves. To save the entire object, specify 0 for offset and 0 for span.

,.span
   Specifies how many 4096-byte blocks CSRSAVE is to save.
   Define span as integer data of length 4.

,.new_hi_offset
   When CSRSAVE completes, new_hi_offset contains the new size of the object expressed in units of 4096 bytes.
   Define new_hi_offset as integer data of length 4.
CSRSAVE callable service

,return_code
When CSRSAVE completes, return_code contains the return code. Define return_code as integer data of length 4.

,reason_code
When CSRSAVE completes, reason_code contains the reason code. Define reason_code as integer data of length 4.

ABEND codes
CSRSAVE might abnormally terminate with abend code X'019'. See System Codes for an explanation and programmer responses.

Return and reason codes
When the CSRSAVE service returns control to your program, GPR 15 (and return_code) contains a return code. GPR 0 (and reason_code) contains a reason code. The following table identifies return code and reason code combinations, and explains their meanings.

A return code of X'4' with a reason code of X'0807' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes, which are two bytes long and right justified, are explained in the description of the DIV macro (Chapter 88, “DIV — Data-in-virtual,” on page 463).

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000</td>
<td>00000000</td>
<td><strong>Meaning:</strong> The operation was successful. <strong>Action:</strong> None.</td>
</tr>
</tbody>
</table>
| 00000004                | xxxx0807               | **Meaning:** Environmental error. Media damage might be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. The value X'0807' is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.  
Action: See the DIV macro description for an explanation of X'0807'. |
| 00000008                | 00000143                | **Meaning:** Program error. You cannot use the SAVE service for a temporary object.  
Action: Call CSRSCOT to save changes made to a temporary object. |
| 0000000C                | xxxxnnnn               | **Meaning:** The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.  
Action: See the DIV macro description for an explanation of nnnn. |
## CSRSAVE callable service

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0000002C                | 00000004                | **Meaning:** Program error. Window services have not been defined to your system, or the link to the service failed.  
**Action:** If window services are available on your system, rerun the program one or more times. If the problem persists, contact the appropriate IBM support personnel. |
Chapter 78. CSRSCOT — Save object changes in a scroll area

Description

Call the CSRSCOT window service to:

- Update specified blocks of a permanent object's scroll area with changes that appear in a window you have defined for the object. CSRSCOT requires that the permanent object have a scroll area. CSRSCOT changes only the content of the scroll area and not the content of the permanent data object.
- Update specified blocks of a temporary data object with the changes that appear in a window you have defined for the data object.

Environment

The requirements for the caller are:

Minimum authorization: Problem state with PSW key 8-15
Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 24- or 31-bit, but all addresses must be 31-bit addresses
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Must be in the primary address space

Programming requirements

None.

Restrictions

The caller must follow all the restrictions imposed by the DIV macro.

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a standard 18-word save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
CSRSCOT callable 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.

Performance implications

None.

Syntax

Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRSCOT
    ,object_id
    ,offset
    ,span
    ,return_code
    ,reason_code)
```

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- `,(object_id`
  - Specifies the object identifier. Supply the same object identifier that CSRIDAC returned when you obtained access to the object.
  - Define `object_id` as character data of length 8.

- `,offset`
  - Specifies the offset into the object in blocks of 4096 bytes. A value of 0 specifies the first block of 4096 bytes or bytes 0 to 4095 of the object; a value of 1 specifies the second block of 4096 bytes, or bytes 4096 to 8191 of the object, and so forth.
  - Define `offset` as integer data of length 4.

  `offset` and `span`, together, determine what part of the object CSRSCOT updates. To update the entire object, specify 0 for `offset` and 0 for `span`.

- `,span`
  - Specifies how many 4096-byte blocks CSRSCOT is to update.
  - Define `span` as integer data of length 4.

- `,return_code`
  - When CSRSCOT completes, `return_code` contains the return code. Define `return_code` as integer data of length 4.

- `,reason_code)`
  - When CSRSCOT completes, `reason_code` contains the reason code. Define `reason_code` as integer data of length 4.

ABEND codes

CSRSCOT might abnormally terminate with abend code X'019'. See z/OS MVS System Codes for an explanation and programmer responses.
### Return and reason codes

When CSRSCOT returns control to your program, GPR 15 (and `return_code`) contains a return code. GPR 0 (and `reason_code`) contains a reason code. The following table identifies return code and reason code combinations and tells what each means.

A return code of X'4' with a reason code of X'0807' or a return code of X'C' with any reason code means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes, which are two bytes long and right justified, are explained in the description of the DIV macro ([Chapter 88, “DIV — Data-in-virtual,” on page 463](#)).

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000 00000000</td>
<td></td>
<td><strong>Meaning</strong>: The operation was successful. <strong>Action</strong>: None.</td>
</tr>
<tr>
<td>00000004 xxxx0807</td>
<td></td>
<td><strong>Meaning</strong>: Environmental error. Media damage might be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE operation completed successfully. The value X'0807' is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface. <strong>Action</strong>: See the DIV macro description for an explanation of X'0807'.</td>
</tr>
<tr>
<td>0000000C xxxxnnnn</td>
<td></td>
<td><strong>Meaning</strong>: The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface. <strong>Action</strong>: See the DIV macro description for an explanation of nnnn.</td>
</tr>
<tr>
<td>0000002C 00000004</td>
<td></td>
<td><strong>Meaning</strong>: Program error or system error. Window services have not been defined to your system, or the link to the service failed. <strong>Action</strong>: If window services are available on your system, rerun the program one or more times. If the problem persists, contact the appropriate IBM support personnel.</td>
</tr>
</tbody>
</table>
CSRSCOT callable service
Chapter 79. CSRSI — System information service

Description

Use the CSRSI service to retrieve system information. You can request information about the machine itself, the logical partition (LPAR) in which the machine is running, or the virtual machine hypervisor (VM) under which the system is running. The returned information is mapped by DSECTs in macro CSRSIIDF (for assembler language callers) or structures in header file CSRSIC (for C language callers).

The information available depends upon the availability of the Store System Information (STSI) instruction. When the STSI instruction is not available (which would be indicated by receiving the return code 4 (equate symbol CSRSI_STSINOTAVAILABLE), only the SI00PCCACPID, SI00PCCACPUA, and SI00PCCACAFM fields within the returned infoarea are valid. When the STSI instruction is available, the validity of the returned infoarea depends upon the system:

- If the system is running neither under LPAR nor VM, then only the CSRSI_Request_V1CPC_Machine data are valid.
- If the system is running under a logical partition (LPAR), then both the CSRSI_Request_V1CPC_Machine data and CSRSI_Request_V2CPC_LPAR data are valid.
- If the system is running under a virtual machine hypervisor (VM), then all of the data (CSRSI_Request_V1CPC_Machine, CSRSI_Request_V2CPC_LPAR, and CSRSI_Request_V3CPC_VM) are valid.

You can request any or all of the information regardless of your system, and validity bits will indicate which returned areas are valid.

Environment

The requirements for the caller are:

Minimum authorization: Problem state, key 8-15
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 24- or 31-bit when using the CALL CSRSI form (or csrsi in C), 31-bit when using an alternate form
ASC mode: Primary
Interrupt status: Enabled or disabled for I/O and external interrupts.
Locks: The caller may hold a LOCAL lock, the CMS lock, or the CPU lock, but is not required to hold any locks.

Programming requirements

The caller should include the CSRSIIDF macro to map the returned information and to provide equates for the service.

Restrictions

None.

Input register information

The caller is not required to set up any registers.
CSRSI callable service

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

CALL CSRSI

In C: the syntax is similar. You can use either of the following techniques to invoke the service:

1. CSRSI (Request,...Returncode);
   When you use this technique, you must link edit your program with a linkage-assist routine (also called a stub) in SYS1.CSSLIB.
2. CSRSI_byaddr (Request,...Returncode);
   Both of these techniques require AMODE=31. If you use the second technique, before you issue the CALL, you must verify that the CSRSI service is available (in the CVT, both CVTOSEXT and CVTCSRSI bits are set on).

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

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

2. L 15,X'10' Get CVT
   L 15,X'220'(.15) Get address of CSRSI
   L 15,X'30'(.15) Get address of CSRSI
   CALL (15),(...)
   Both of these techniques require AMODE=31. If you use the second technique, before you issue the CALL, you must verify that the CSRSI service is available (in the CVT, both CVTOSEXT and CVTCSRSI bits are set on).

Parameters

The parameters are explained as follows:

(Request
   Supplied parameter:
   • Type: Integer
   • Length: Full word
CSRSI callable service

Request identifies the type of system information to be returned. The field must contain a value that represents one or more of the possible request types. You add the values to create the full word. Do not specify a request type more than once. The possible request types, and their meanings, are:

- **CSRSI_Request_V1CPC_Machine**
  The system is to return information about the machine.

- **CSRSI_Request_V2CPC_LPAR**
  The system is to return information about the logical partition (LPAR).

- **CSRSI_Request_V3CPC_VM**
  The system is to return information about the virtual machine (VM).

,Infoarealen
  Supplied parameter:
  - Type: Integer
  - Range: X'1040', X'2040', X'3040', X'4040'
  - Length: Full word

Infoarealen specifies the length of the infoarea parameter.

,Infoarea
  Returned parameter:
  - Type: Character
  - Length: X'1040', X'2040', X'3040', X'4040' bytes

Infoarea is to contain the retrieved system information. (Infoarealen specifies the length of the provided area.) The infoarea must be of the proper length to hold the requested information. This length depends on the value of the Request parameter.

- When the Request parameter is CSRSI_Request_V1CPC_Machine, the returned infoarea is mapped by SIV1 and the infoarealen parameter must be X'2040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V2CPC_LPAR, the returned infoarea is mapped by SIV1V2 and the infoarealen parameter must be X'3040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V2CPC_LPAR plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV1V2V3 and the infoarealen parameter must be X'4040'.
- When the Request parameter is CSRSI_Request_V1CPC_Machine plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV1V3 and the infoarealen parameter must be X'3040'.
- When the Request parameter is CSRSI_Request_V2CPC_LPAR, the returned infoarea is mapped by SIV2 and the infoarealen parameter must be X'1040'.
- When the Request parameter is CSRSI_Request_V2CPC_LPAR plus CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV2V3 and the infoarealen parameter must be X'2040'.
- When the Request parameter is CSRSI_Request_V3CPC_VM, the returned infoarea is mapped by SIV3 and the infoarealen parameter must be X'1040'.

,Returncode
  Returned parameter:
  - Type: Integer
  - Length: Full word

Returncode contains the return code from the CSRSI service.
CSRSI callable service

Return codes

When the CSRSI service returns control to the caller, Returncode contains the return code. To obtain the equates for the return codes:

- If you are coding in assembler, include mapping macro `CSRSIIDF`, described in `z/OS MVS Data Areas` in z/OS Internet Library at [http://www.ibm.com/systems/z/os/zos/bkserv/].
- If you are coding in C, use include file `CSRSIC`. See Figure 4.

The following table describes the return codes, shown in decimal.

<table>
<thead>
<tr>
<th>Return Code (decimal)</th>
<th>Equate Symbol Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Equate Symbol: CSRSI_SUCCESS</td>
</tr>
<tr>
<td><strong>Meaning:</strong></td>
<td>The CSRSI service completed successfully. All information requested was returned.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Check the si00validityflags field to determine the validity of each returned area.</td>
</tr>
<tr>
<td>04</td>
<td>Equate Symbol: CSRSI_STSINOTAVAILABLE</td>
</tr>
<tr>
<td><strong>Meaning:</strong></td>
<td>The CSRSI service completed successfully, but since the Store System Information (STSI) instruction was not available, only the SI00PCCACPID, SI00PCCACPUA, and SI00PCCACAFM fields are valid.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>None required.</td>
</tr>
<tr>
<td>08</td>
<td>Equate Symbol: CSRSI_SERVICENOTAVAILABLE</td>
</tr>
<tr>
<td><strong>Meaning:</strong></td>
<td>Environmental error: The CSRSI service is not available on this system.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Avoid calling the CSRSI service unless running on a system on which it is available.</td>
</tr>
<tr>
<td>12</td>
<td>Equate Symbol: CSRSI_BADREQUEST</td>
</tr>
<tr>
<td><strong>Meaning:</strong></td>
<td>User error: The request parameter did not specify a word formed from any combination of CSRSI_Request_V1CPC_Machine, CSRSI_Request_V2CPC_LPAR, and CSRSI_Request_V3CPC_VM.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Correct the parameter.</td>
</tr>
<tr>
<td>16</td>
<td>Equate Symbol: CSRSI_BADINFOAREALEN</td>
</tr>
<tr>
<td><strong>Meaning:</strong></td>
<td>User error: The Infoarealen parameter did not match the length of the area required to return the requested information.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Correct the parameter.</td>
</tr>
<tr>
<td>20</td>
<td>Equate Symbol: CSRSI_BADLOCK</td>
</tr>
<tr>
<td><strong>Meaning:</strong></td>
<td>User error: The service was called while holding a system lock other than CPU: LOCAL/CML, or CMS.</td>
</tr>
<tr>
<td><strong>Action:</strong></td>
<td>Avoid calling in this environment.</td>
</tr>
</tbody>
</table>
CSRSIC C/370 header file

For a C programmer, include file CSRSIC provides equates for return codes and data constants, such as Register service request types. To use CSRSIC, copy the file from SYS1.SAMPLIB to the appropriate local C library. The contents of the file are displayed in Figure 4.

```c
#ifndef __CSRSI
#define __CSRSI

/*******************************
 * Type Definitions for User Specified Parameters *
 *******************************

/* Type for Request operand of CSRSI */
typedef int CSRSIRequest;

/* Type for InfoAreaLen operand of CSRSI */
typedef int CSRSIInfoAreaLen;

/* Type for Return Code */
typedef int CSRSIReturnCode;

/*******************************
 * Function Prototypes for Service Routines *
 *******************************

#ifdef __cplusplus
extern "OS" ??&>
#else
#pragma linkage(CSRSI_calltype,OS)
#endif
typedef void CSRSI_calltype(
    CSRSIRequest __REQUEST, /* Input - request type */
    CSRSIInfoAreaLen __INFOAREALEN, /* Input - length of infoarea */
    void *__INFOAREA, /* Input - info area */
    CSRSIReturnCode *__RC); /* Output - return code */

extern CSRSI_calltype csrsi;

#endif
#endif

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 1 of 14)
CSRSI callable service

```c
struct CSRSI_CVT
{
    unsigned char CSRSI_cvt_filler1; /* (116??) */

    struct
    {
        int CSRSI_cvtdcb_rsvd1:4; /* Not needed */
        int CSRSI_cvtosext:1; /* If on, indicates that the
                                CVTOSLV fields are valid */
        int CSRSI_cvtdcb_rsvd2:3; /* Not needed */
    } CSRSI_cvtdcb;

    unsigned char CSRSI_cvt_filler2; /* (427??) */

    struct CSRSI_CSRT * CSRSI_cvtcsrt;

    unsigned char CSRSI_cvt_filler3; /* (716??) */

    unsigned char CSRSI_cvtoslv0;
    unsigned char CSRSI_cvtoslv1;
    unsigned char CSRSI_cvtoslv2;
    unsigned char CSRSI_cvtoslv3;

    struct
    {
        int CSRSI_cvtsrsi:1; /* If on, indicates that the
                                CSRSI service is available */
        int CSRSI_cvtoslv1_rsvd1:7; /* Not needed */
    } CSRSI_cvtoslv4;

    unsigned char CSRSI_cvt_filler4; /* (11??) */
};

struct CSRSI_PSA
{
    char CSRSI_psa_filler; /* (16??) */

    struct CSRSI_CVT* CSRSI_cvt;
};

/* End of CSRSI Header */
#endif

********************************************************************
/* si11v1 represents the output for a V1 CPC when general CPC */
/* information is requested */
********************************************************************

typedef struct
{
    unsigned char _filler1; /* (32??) */ /* Reserved */
    unsigned char si11v1pcmanufacturer; /* (16??) */ /* The 16-character (0-9
                                            or uppercase A-Z) EBCDIC name
                                            of the manufacturer of the V1
                                            CPC. The name is left-justified with trailing
                                            blank characters if necessary. */
    unsigned char si11v1ctype; /* (4??) */ /* The 4-character (0-9) EBCDIC
                                            type identifier of the V1 CPC. */
    unsigned char _filler2; /* (12??) */ /* Reserved */
} si11v1cpc;

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 2 of 14)
unsigned char s11v1cpcmodel??(16??); /* The 16-character (0-9 or uppercase A-Z) EBCDIC model identifier of the V1 CPC. The identifier is left-justified with trailing blank characters if necessary. */

unsigned char s11v1cpcsequencecode??(16??); /* The 16-character (0-9 or uppercase A-Z) EBCDIC sequence code of the V1 CPC. The sequence code is right-justified with leading EBCDIC zeroes if necessary. */

unsigned char s11v1cpcplantofmanufacture??(4??); /* The 4-character (0-9 or uppercase A-Z) EBCDIC plant code that identifies the plant of manufacture for the V1 CPC. The plant code is left-justified with trailing blank characters if necessary. */

unsigned char _filler3??(3996??); /* Reserved */

//si11v1;

/* si22v1 represents the output for a V1 CPC when information is requested about the set of CPUs */
typedef struct ??&>
unsigned char _filler1??(32??); /* Reserved */
unsigned char si22v1cpucapability??(4??); /* An unsigned binary integer that specifies the capability of one of the CPUs contained in the V1 CPC. It is used as an indication of the capability of the CPU relative to the capability of other CPU models. */

unsigned int si22v1totalcpucount : 16; /* A 2-byte unsigned integer that specifies the total number of CPUs contained in the V1 CPC. This number includes all CPUs in the configured state, the standby state, and the reserved state. */

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 3 of 14)
unsigned int si22v1configuredcpucount : 16; /* A 2-byte unsigned binary integer that specifies the total number of CPUs that are in the configured state. A CPU is in the configured state when it is described in the V1-CPC configuration definition and is available to be used to execute programs. */

unsigned int si22v1standbycpucount : 16; /* A 2-byte unsigned integer that specifies the total number of CPUs that are in the standby state. A CPU is in the standby state when it is described in the V1-CPC configuration definition, is not available to be used to execute programs, but can be used to execute programs by issuing instructions to place it in the configured state. */

unsigned int si22v1reservedcpucount : 16; /* A 2-byte unsigned binary integer that specifies the total number of CPUs that are in the reserved state. A CPU is in the reserved state when it is described in the V1-CPC configuration definition, is not available to be used to execute programs, and cannot be made available to be used to execute programs by issuing instructions to place it in the configured state, but it may be possible to place it in the standby or configured state through manually initiated actions */

struct ??&>
    unsigned char _si22v1mpcpucapaf??(2??); /* Each individual adjustment factor. */
    unsigned char _filler2??(4050??);
?> si22v1mpcpucapafs;
?> si22v1;
#define si22v1mpcpucapaf si22v1mpcpucapafs._si22v1mpcpucapaf

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 4 of 14)
typedef struct
unsigned char _filler1[(32)]; /* Reserved */
unsigned int _si22v2cpcnumber : 16; /* A 2-byte unsigned integer which is the number of this V2 CPC. This number distinguishes this V2 CPC from all other V2 CPCs provided by the same logical-partition hypervisor */
unsigned char _filler2; /* Reserved */
struct
unsigned int _si22v2lcpudedicated : 1; /* When one, indicates that one or more of the logical CPUs for this V2 CPC are provided using V1 CPUs that are dedicated to this V2 CPC and are not used to provide logical CPUs for any other V2 CPCs. The number of logical CPUs that are provided using dedicated V1 CPUs is specified by the dedicated-LCPU-count value. When zero, bit 0 indicates that none of the logical CPUs for this V2 CPC are provided using V1 CPUs that are dedicated to this V2 CPC. */
unsigned int _si22v2lcpushared : 1; /* When one, indicates that one or more of the logical CPUs for this V2 CPC are provided using V1 CPUs that can be used to provide logical CPUs for other V2 CPCs. The number of logical CPUs that are provided using shared V1 CPUs is specified by the shared-LCPU-count value. When zero, it indicates that none of the logical CPUs for this V2 CPC are provided using shared V1 CPUs. */

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 5 of 14)
unsigned int _si22v2lcpuulimit : 1; /*
Utilization limit. When one, indicates that the amount of
use of the V1-CPC CPUs that are used to provide the
logical CPUs for this V2 CPC is limited. When zero, it
indicates that the amount of
use of the V1-CPC CPUs that are used to provide the
logical CPUs for this V2 CPC is unlimited. */

unsigned int _filler3 : 5; /* Reserved */

??> si22v21cpu; /* Characteristics */

unsigned int si22v2totalcpucount : 16; /*
A 2-byte unsigned
integer that specifies the
total number of logical CPUs that are provided for this V2
CPC. This number includes all
of the logical CPUs that are
in the configured state, the
standby state, and the
reserved state. */

unsigned int si22v2configuredcpucount : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs for this V2 CPC that are
in the configured state. A
logical CPU is in the
configured state when it is
described in the V2-CPC
configuration definition and
is available to be used to
execute programs. */

unsigned int si22v2standbyclpucount : 16; /*
A 2-byte unsigned
binary integer that specifies
the total number of logical
CPUs that are in the standby
state. A logical CPU is in the
standby state when it is
described in the V2-CPC
configuration definition, is
not available to be used to
execute programs, but can be
used to execute programs by
issuing instructions to place
it in the configured state.

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 6 of 14)
### CSRSI callable service

```c
unsigned int si22v2reservedlcpucount : 16; /*
   A 2-byte unsigned binary integer that specifies the total number of logical CPUs that are in the reserved state. A logical CPU is in the reserved state when it is described in the V2-CPC configuration definition, is not available to be used to execute programs, and cannot be made available to be used to execute programs by issuing instructions to place it in the configured state, but it may be possible to place it in the standby or configured state through manually initiated actions */

unsigned char si22v2cpcname[16]; /*
   The 8-character EBCDIC name of this V2 CPC. The name is left-justified with trailing blank characters if necessary. */

unsigned char si22v2cpccapabilityaf[4]; /* Capability Adjustment Factor (CAF). An unsigned binary integer of 1000 or less. The adjustment factor specifies the amount of the V1-CPC capability that is allowed to be used for this V2 CPC by the logical-partition hypervisor. The fraction of V1-CPC capability is determined by dividing the CAF value by 1000. */

unsigned char _filler4[16]; /* Reserved */

unsigned int si22v2dedicatedlcpucount : 16; /*
   A 2-byte unsigned binary integer that specifies the number of configured-state logical CPUs for this V2 CPC that are provided using dedicated V1 CPUs. (See the description of bit si22v21cpudirected.) */
```

*Figure 4. CSRSIC from SYS1.SAMPLIB (Part 7 of 14)*
unsigned int si22v2sharedlpucount : 16; /*
   A 2-byte unsigned integer that specifies the number of configured-state logical CPUs for this V2 CPC that are provided using shared V1 CPUs. (See the description of bit si22v2lcpushared.) */

unsigned char _filler5??(4012??); /* Reserved */
?
?? si22v2;

#define si22v2lcpudedicated si22v2lcpuc._si22v2lcpudedicated
#define si22v2lcpushared si22v2lcpuc._si22v2lcpushared
#define si22v2lcpuulimit si22v2lcpuc._si22v2lcpuulimit

/*******************************************************************************
/* si22v3db is a description block that comprises part of the */
/* si22v3 data. */
/*******************************************************************************

typedef struct ??&>
unsigned char _filler1??(4??); /* Reserved */
unsigned int si22v3dbtotallpucount : 16; /*
   A 2-byte unsigned binary integer that specifies the total number of logical CPUs that are provided for this V3 CPC. This number includes all of the logical CPUs that are in the configured state, the standby state, and the reserved state. */

unsigned int si22v3dbconfiguredlpucount : 16; /*
   A 2-byte unsigned binary integer that specifies the number of logical CPUs for this V3 CPC that are in the configured state. A logical CPU is in the configured state when it is described in the V3-CPC configuration definition and is available to be used to execute programs. */

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 8 of 14)
unsigned int si22v3dbstandbylcpucount : 16; /*
   A 2-byte unsigned binary integer that specifies the number of logical CPUs for this V3 CPC that are in the standby state. A logical CPU is in the standby state when it is described in the V3-CPC configuration definition, is not available to be used to execute programs, but can be used to execute programs by issuing instructions to place it in the configured state. */

unsigned int si22v3dbreservedlcpucount : 16; /*
   A 2-byte unsigned binary integer that specifies the number of logical CPUs for this V3 CPC that are in the reserved state. A logical CPU is in the reserved state when it is described in the V2-CPC configuration definition, is not available to be used to execute programs, and cannot be made available to be used to execute programs by issuing instructions to place it in the configured state, but it may be possible to place it in the standby or configured state through manually initiated actions. */

unsigned char si22v3dbcpcname??(8??); /* The 8-character EBCDIC name of this V3 CPC. The name is left-justified with trailing blank characters if necessary. */

unsigned char si22v3dbcpccaf??(4??); /* A 4-byte unsigned binary integer that specifies an adjustment factor. The adjustment factor specifies the amount of the V1-CPC or V2-CPC capability that is allowed to be used for this V3 CPC by the virtual-machine-hypervisor program. */

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 9 of 14)
CSRSI callable service

```c
unsigned char si22v3dbvmhpidentifier[16]; /* The 16-character EBCDIC identifier of the virtual-machine-hypervisor program that provides this V3 CPC. (This identifier may include qualifiers such as version number and release level). The identifier is left-justified with trailing blank characters if necessary. */

unsigned char _filler2[24]; /* Reserved */
	/*********************************************************************/
/* si22v3 represents the output for a V3 CPC when information */
/* is requested about the set of CPUs */
	/*********************************************************************/
typedef struct {?
unsigned char _filler1[28]; /* Reserved */
unsigned char _filler2[3]; /* Reserved */
struct {?
    unsigned int _filler3 : 4; /* Reserved */
    unsigned int _si22v3dbcount : 4; /* Description Block Count. A 4-bit unsigned binary integer that indicates the number (up to 8) of V3-CPC description blocks that are stored in the si22v3dbe array. */
} si22v3dbcountfield; /* */
} si22v3db; /* Array of entries. Only the number indicated by si22v3dbcount are valid */

unsigned char _filler5[3552]; /* Reserved */
	/*********************************************************************/
/* SI00 represents the "starter" information. This structure is */
/* part of the information returned on every CSRSI request. */
	/*********************************************************************/

#define si22v3dbcount si22v3dbcountfield._si22v3dbcount
	/*********************************************************************/
/* Figure 4. CSRSIC from SYS1.SAMPLIB (Part 10 of 14) */
```
typedef struct {
    char sio0cpcvariety; /* SI00CPCVariety_V1CPC_MACHINE, SI00CPCVariety_V2CPC_LPAR, or SI00CPCVariety_V3CPC_VM */
}

struct {
    int _sio0validsi11v1 : 1; /* si11v1 was requested and the information returned is valid */
    int _sio0validsi22v1 : 1; /* si22v1 was requested and the information returned is valid */
    int _sio0validsi22v2 : 1; /* si22v2 was requested and the information returned is valid */
    int _sio0validsi22v3 : 1; /* si22v3 was requested and the information returned is valid */
    int _filler1 : 4; /* Reserved */
    unsigned char _filler2 : 2; /* Reserved */
    unsigned char sio0pccacpid : 12; /* PCCACPID value for this CPU */
    unsigned char sio0pccacpu : 2; /* PCCACPUA value for this CPU */
    unsigned char sio0pccacafm : 2; /* PCCACAFM value for this CPU */
    unsigned char _filler3 : 4; /* Reserved */
    unsigned char sio0lastupdatetimestamp : 8; /* Time of last STSI update, via STCK */
    unsigned char _filler4 : 32; /* Reserved */
} sio0;

#define sio0validsi11v1 sio0validityflags._sio0validsi11v1
def sio0validsi22v1 sio0validityflags._sio0validsi22v1
def sio0validsi22v2 sio0validityflags._sio0validsi22v2
def sio0validsi22v3 sio0validityflags._sio0validsi22v3

typedef struct {
    sio0 siv1sio0; /* Area mapped by */
    struct sio0
        sio11v1 siv1sio11v1; /* Area */
        sio12v1 siv1sio12v1; /* Area */
} siv1;

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 11 of 14)
typedef struct {
  sio0 siv1v2sio0; /* Area mapped by */
  sii1v1 siv1v2sii1v1; /* Area */
  sii2v1 siv1v2si2v1; /* Area */
  sii2v2 siv1v2si2v2; /* Area */
} siv1v2;

typedef struct {
  sio0 siv1v2v3sio0; /* Area */
  sii1v1 siv1v2v3sii1v1; /* Area */
  sii2v1 siv1v2v3si2v1; /* Area */
  sii2v2 siv1v2v3si2v2; /* Area */
  sii2v3 siv1v2v3si2v3; /* Area */
} siv1v2v3;

typedef struct {
  sio0 siv1v3sio0; /* Area */
  sii1v1 siv1v3sii1v1; /* Area */
  sii2v1 siv1v3si2v1; /* Area */
  sii2v3 siv1v3si2v3; /* Area */
} siv1v3;

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 12 of 14)
Figure 4. CSRSIC from SYS1.SAMPLIB (Part 13 of 14)
/** Fixed Service Parameter and Return Code Defines */
/** SI00 Constants */
#define SI00CPCVARIETY_V1CPC_MACHINE 1
#define SI00CPCVARIETY_V2CPC_LPAR 2
#define SI00CPCVARIETY_V3CPC_VM 3
/** CSRSI Constants */
#define CSRSI_REQUEST_V1CPC_MACHINE 1
#define CSRSI_REQUEST_V2CPC_LPAR 2
#define CSRSI_REQUEST_V3CPC_VM 4
/** CSRSI Return codes */
#define CSRSI_SUCCESS 0
#define CSRSI_STSINOTAVAILABLE 4
#define CSRSI_SERVICENOTAVAILABLE 8
#define CSRSI_BADREQUEST 12
#define CSRSI_BADINFOAREALEN 16
#define CSRSI_BADLOCK 20

Figure 4. CSRSIC from SYS1.SAMPLIB (Part 14 of 14)
Chapter 80. CSRUNIC — Unicode instruction services

Description

CSRUNIC allows you to request processing of a group of instructions related to Unicode data. Unicode data uses the binary codes of the Unicode Worldwide Character Standard; these codes support the characters of most of the world's written languages. For details about the Unicode instructions, see z/Architecture Principles of Operations, SA22-7832. The CSRUNIC macro invokes the requested instructions by name, if the Unicode hardware is present. If the hardware is not present, the macro simulates the requested instructions.

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state and PSW key 8-15
- **Dispatchable unit mode:** Task
- **Cross memory mode:** Any PASN, any HASN, any SASN
- **AMODE:** 31-bit
- **ASC mode:** Primary or access register (AR)
- **Interrupt status:** Enabled or disabled for I/O and external interrupts
- **Locks:** The caller is not required to hold any locks on entry. The caller may hold the local, CMS, or CPU lock.
- **Control parameters:** None.

Programming requirements

The caller must include the CSRYUNIC macro to get a mapping for the parameter block for the requested function. The CSRYUNIC macro also includes symbolic equates for the return codes from the service.

Restrictions

None.

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>Address of standard 72-byte save area. When not in AR-ASC mode, the area must be in the primary address space. When in AR-ASC mode, it must be in the space addressed via the ALET in access register 13.</td>
</tr>
</tbody>
</table>

Before issuing the CSRUNIC macro in AR-ASC mode, the caller must ensure that the following access registers (ARs) contain the specified information:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>ALET of the 72-byte save area pointed to by GPR 13.</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The CSRUNIC macro is written as follows:

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

b

CSRUNIC

b
```

```
FUNCTION=MVCLU
FUNCTION=CLCLU
FUNCTION=TP
FUNCTION=PKA
FUNCTION=PKU
FUNCTION=UNPKA
FUNCTION=UNPKU
FUNCTION=TRTT
FUNCTION=TRTO
FUNCTION=TROT
FUNCTION=TROO
FUNCTION=TRE
FUNCTION=CUUTF
FUNCTION=CUTFU

,PBLOCK=pblock pblock: RX-type address or address in register (2) - (12).

,RETCODE=retcode retcode: RS-type address or register (2) - (12).
```
Parameters

The parameters are explained as follows:

*name*

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

```c
FUNCTION=MVCLU
FUNCTION=CLCLU
FUNCTION=TP
FUNCTION=PKA
FUNCTION=PKU
FUNCTION=UNPKA
FUNCTION=UNPKU
FUNCTION=TRTT
FUNCTION=TRTO
FUNCTION=TROT
FUNCTION=TROO
FUNCTION=TRE
FUNCTION=CUTF
FUNCTION=CUTFU
```

A required parameter that designates the function to be performed.

```c
FUNCTION=MVCLU
  indicates to process an MVCLU operation.
FUNCTION=CLCLU
  indicates to process a CLCLU operation.
FUNCTION=TP
  indicates to process a TP operation.
FUNCTION=PKA
  indicates to process a PKA operation.
FUNCTION=PKU
  indicates to process a PKU operation.
FUNCTION=UNPKA
  indicates to process an UNPKA operation.
FUNCTION=UNPKU
  indicates to process an UNPKU operation.
FUNCTION=TRTT
  indicates to process a TRTT operation.
FUNCTION=TRTO
  indicates to process a TRTO operation.
FUNCTION=TROT
  indicates to process a TROT operation.
FUNCTION=TROO
  indicates to process a TROO operation.
FUNCTION=TRE
  indicates to process a TRE operation.
```
CSRUNIC macro

**FUNCTION=CUUTF**
indicates to process a CUUTF operation.

**FUNCTION=CUTFU**
indicates to process a CUTFU operation.

**PBLCK=pblock**
A required input parameter, area that is mapped by DSECTs in macro CSRYUNIC that correlate to the function requested. The area provides the information needed by, and provided on return by, the CSRUNIC service. It should begin on a fullword boundary.

The name of the DSECT is "UNIC_" followed by the requested function (for example, UNIC_MVCLU for the MVCLU function).

**To code:** Specify the RX-type address, or address in register (2) - (12), of a 36-character field.

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

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

**ABEND codes**

0C4 The user may get this completion code if a user-provided data area is not accessible.

0C6 The user may get this completion code if the instruction has been executed in the hardware and the provided data does not meet the requirements for that instruction.
  - For MVCLU, either the source length or target length was not even.
  - For CLCLU, either the source length or target length was not even.
  - For PKA, the source length exceeded 31.
  - For PKU, the source length exceeded 64 or was not even (that is, the LengthMinusOne was not odd).
  - For UNPKA, the target length exceeded 31.
  - For UNPKU, the target length exceeded 64 or was not even (that is, the LengthMinusOne was not odd).
  - For TRTT, the length was not even.
  - For TRTO, the length was not even.
  - For CUTFU, a bad UTF-8 character was encountered.

The user may get this completion code if the work area was not on a doubleword boundary.

**Return codes**

When the CSRUNIC macro returns control to your program, GPR 15 (and retcode, when you code RETCODE) contains a return code.

Return code constants are defined in macro CSRYUNIC.

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code.
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0           | UNIC_MVCLU_RC_OpLengthsEqual | **Meaning**: The operand lengths were the same.  
**Action**: None required. |
| 4           | UNIC_MVCLU_RC_TargetLengthShorter | **Meaning**: The target operand was shorter than the source operand.  
**Action**: None required. |
| 8           | UNIC_MVCLU_RC_TargetLengthLonger | **Meaning**: The target operand was longer than the source operand.  
**Action**: None required. |
| 10          | UNIC_MVCLU_RC_TargetLengthNotEven | **Meaning**: The target operand was not an even number of bytes.  
**Action**: Only call CSRUNIC FUNCTION=MVCLU when the target operand is an even number of bytes (that is, a whole number of unicode characters). |
| 14          | UNIC_MVCLU_RC_SourceLengthNotEven | **Meaning**: The source operand was not an even number of bytes.  
**Action**: Only call CSRUNIC FUNCTION=MVCLU when the source operand is an even number of bytes (that is, a whole number of unicode characters). |
| 1C          | UNIC_MVCLU_RC_WorkareaNotAligned | **Meaning**: The workarea provided was not on a doubleword boundary.  
**Action**: Make sure that the workarea is on a doubleword boundary. |
| 0           | UNIC_CLCLU_RC_OperandsEqual | **Meaning**: the two operands were equal.  
**Action**: None required. |
| 4           | UNIC_CLCLU_RC_LeftOpLessThanRight | **Meaning**: The left operand was less than the right operand.  
**Action**: None required. |
| 8           | UNIC_CLCLU_RC_RightOpLessThanLeft | **Meaning**: The right operand was less than the left operand.  
**Action**: None required. |
| 10          | UNIC_CLCLU_RC_LeftOpLengthNotEven | **Meaning**: The left operand was not an even number of bytes.  
**Action**: Only call CSRUNIC FUNCTION=CLCLU when the left operand is an even number of bytes (that is, a whole number of unicode characters). |
| 14          | UNIC_CLCLU_RC_RightOpLengthNotEven | **Meaning**: The right operand was not an even number of bytes.  
**Action**: Only call CSRUNIC FUNCTION=CLCLU when the right operand is an even number of bytes (that is, a whole number of unicode characters). |
### Table 11. Return Codes for the CSRUNIC Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1C</td>
<td>UNIC_CLCLU_RC_WorkareaNotAligned</td>
<td><strong>Meaning:</strong> The workarea provided was not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the workarea is on a doubleword boundary.</td>
</tr>
<tr>
<td>0</td>
<td>UNIC_TP_RC_Valid</td>
<td><strong>Meaning:</strong> the operand is a valid packed number.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>4</td>
<td>UNIC_TP_RC_SignNotValid</td>
<td><strong>Meaning:</strong> The sign of the operand was not valid. All the digits were valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>8</td>
<td>UNIC_TP_RC_DigitNotValid</td>
<td><strong>Meaning:</strong> One or more digits of the operand were not valid. The sign was valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>0C</td>
<td>UNIC_TP_RC_SignDigitNotValid</td>
<td><strong>Meaning:</strong> The sign and one or more digits of the operand were not valid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>1C</td>
<td>UNIC_TP_RC_WorkareaNotAligned</td>
<td><strong>Meaning:</strong> The workarea provided was not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the workarea is on a doubleword boundary.</td>
</tr>
<tr>
<td>0</td>
<td>UNIC_PKA_RC_OK</td>
<td><strong>Meaning:</strong> The pack operation completed successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>14</td>
<td>UNIC_PKA_RC_SourceLengthNotValid</td>
<td><strong>Meaning:</strong> The length of the source operand exceeded 32 bytes (that is, the LengthMinusOne exceeded 31).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Avoid calling CSRUNIC REQUEST=PKA for an operand longer than 32 bytes.</td>
</tr>
<tr>
<td>1C</td>
<td>UNIC_PKA_RC_WorkareaNotAligned</td>
<td><strong>Meaning:</strong> The workarea provided was not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the workarea is on a doubleword boundary.</td>
</tr>
<tr>
<td>0</td>
<td>UNIC_PKU_RC_OK</td>
<td><strong>Meaning:</strong> The pack operation completed successfully.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>14</td>
<td>UNIC_PKU_RC_SourceLengthNotValid</td>
<td><strong>Meaning:</strong> The length of the source operand exceeded 64 bytes (that is, the LengthMinusOne exceeded 63).</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Avoid calling CSRUNIC REQUEST=PKU for an operand longer than 64 bytes.</td>
</tr>
<tr>
<td>Return Code</td>
<td>Equate Symbol</td>
<td>Meaning and Action</td>
</tr>
<tr>
<td>-------------</td>
<td>----------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 24          | UNIC_PKU_RC_SourceLengthNotEven        | **Meaning**: The source operand was not an even number of bytes.  
**Action**: Only call CSRUNIC FUNCTION=PKU when the source operand is an even number of bytes (that is, a whole number of unicode characters). |
| 1C          | UNIC_PKU_RC_WorkareaNotAligned         | **Meaning**: The workarea provided was not on a doubleword boundary.  
**Action**: Make sure that the workarea is on a doubleword boundary. |
| 0           | UNIC_UNPKA_RC_Positive                 | **Meaning**: The operand represented a positive number.  
**Action**: None required. |
| 4           | UNIC_UNPKA_RC_Negative                 | **Meaning**: The operand represented a negative number.  
**Action**: None required. |
| 0C          | UNIC_UNPKA_RC_BadSign                  | **Meaning**: The operand did not have a valid sign.  
**Action**: None required. |
| 14          | UNIC_UNPKA_RC_TargetLengthNotValid     | **Meaning**: The length of the target operand exceeded 32 bytes (that is, the LengthMinusOne exceeded 31).  
**Action**: Avoid calling CSRUNIC REQUEST=PKA for an operand longer than 32 bytes. |
| 1C          | UNIC_UNPKA_RC_WorkareaNotAligned       | **Meaning**: The workarea provided was not on a doubleword boundary.  
**Action**: Make sure that the workarea is on a doubleword boundary. |
| 0           | UNIC_UNPKU_RC_Positive                 | **Meaning**: The operand represented a positive number.  
**Action**: None required. |
| 4           | UNIC_UNPKU_RC_Negative                 | **Meaning**: The operand represented a negative number.  
**Action**: None required. |
| 0C          | UNIC_UNPKU_RC_BadSign                  | **Meaning**: The operand did not have a valid sign.  
**Action**: None required. |
| 14          | UNIC_UNPKU_RC_TargetLengthNotValid     | **Meaning**: The length of the target operand exceeded 64 bytes (that is, the LengthMinusOne exceeded 63).  
**Action**: Avoid calling CSRUNIC REQUEST=PKU for an operand longer than 64 bytes. |
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 24          | UNIC_UNPKU_RC_TargetLengthNotEven | **Meaning:** The target operand was not an even number of bytes.  
**Action:** Only call CSRUNIC FUNCTION=UNPKU when the target operand is an even number of bytes (that is, a whole number of unicode characters). |
| 1C          | UNIC_UNPKU_RC_WorkareaNotAligned | **Meaning:** The workarea provided was not on a doubleword boundary.  
**Action:** Make sure that the workarea is on a doubleword boundary. |
| 0           | UNIC_TRTT_RC_TestCharNotFound | **Meaning:** The translation completed. The test character was not found.  
**Action:** None required. |
| 4           | UNIC_TRTT_RC_TestCharFound | **Meaning:** The test character was found. The operation ended at that point.  
**Action:** None required. |
| 10          | UNIC_TRTT_RC_LengthNotEven | **Meaning:** The operand was not an even number of bytes.  
**Action:** Only call CSRUNIC FUNCTION=TRTT when the operand is an even number of bytes (that is, a whole number of unicode characters). |
| 1C          | UNIC_TRTT_RC_WorkareaNotAligned | **Meaning:** The workarea provided was not on a doubleword boundary.  
**Action:** Make sure that the workarea is on a doubleword boundary. |
| 0           | UNIC_TRTO_RC_TestCharNotFound | **Meaning:** The translation completed. The test character was not found.  
**Action:** None required. |
| 4           | UNIC_TRTO_RC_TestCharFound | **Meaning:** The test character was found. The operation ended at that point.  
**Action:** None required. |
| 10          | UNIC_TRTO_RC_LengthNotEven | **Meaning:** The operand was not an even number of bytes.  
**Action:** Only call CSRUNIC FUNCTION=TRTO when the operand is an even number of bytes (that is, a whole number of unicode characters). |
| 1C          | UNIC_TRTO_RC_WorkareaNotAligned | **Meaning:** The workarea provided was not on a doubleword boundary.  
**Action:** Make sure that the workarea is on a doubleword boundary. |
<table>
<thead>
<tr>
<th>Return Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>UNIC_TROT_RC_TestCharFound</td>
<td><strong>Meaning:</strong> The test character was found. The operation ended at that point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>1C</td>
<td>UNIC_TROT_RC_WorkareaNotAligned</td>
<td><strong>Meaning:</strong> The workarea provided was not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the workarea is on a doubleword boundary.</td>
</tr>
<tr>
<td>0</td>
<td>UNIC_TROO_RC_TestCharNotFound</td>
<td><strong>Meaning:</strong> The translation completed. The test character was not found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>4</td>
<td>UNIC_TROO_RC_TestCharFound</td>
<td><strong>Meaning:</strong> The test character was found. The operation ended at that point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>1C</td>
<td>UNIC_TROO_RC_WorkareaNotAligned</td>
<td><strong>Meaning:</strong> The workarea provided was not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the workarea is on a doubleword boundary.</td>
</tr>
<tr>
<td>0</td>
<td>UNIC_TRE_RC_TestCharNotFound</td>
<td><strong>Meaning:</strong> The translation completed. The test character was not found.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>4</td>
<td>UNIC_TRE_RC_TestCharFound</td>
<td><strong>Meaning:</strong> The test character was found. The operation ended at that point.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>1C</td>
<td>UNIC_TRE_RC_WorkareaNotAligned</td>
<td><strong>Meaning:</strong> The workarea provided was not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the workarea is on a doubleword boundary.</td>
</tr>
<tr>
<td>0</td>
<td>UNIC_CUUTF_RC_SourceExhausted</td>
<td><strong>Meaning:</strong> All unicode characters in the source were converted to their UTF-8 equivalents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
<tr>
<td>4</td>
<td>UNIC_CUUTF_RC_TargetExhausted</td>
<td><strong>Meaning:</strong> The target operand did not have enough room to hold the UTF-8 equivalents of all of the source unicode characters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Provide a larger target area.</td>
</tr>
<tr>
<td>1C</td>
<td>UNIC_CUUTF_RC_WorkareaNotAligned</td>
<td><strong>Meaning:</strong> The workarea provided was not on a doubleword boundary.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> Make sure that the workarea is on a doubleword boundary.</td>
</tr>
<tr>
<td>0</td>
<td>UNIC_CUTFU_RC_SourceExhausted</td>
<td><strong>Meaning:</strong> All UTF-8 characters in the source were converted to their unicode equivalents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Action:</strong> None required.</td>
</tr>
</tbody>
</table>
Table 11. Return Codes for the CSRUNIC Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Equate Symbol</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 4           | UNIC_CUTFU_RC_TargetExhausted | **Meaning:** The target operand did not have enough room to hold the unicode equivalents of all of the source UTF-8 characters.  
**Action:** Provide a larger target area. |
| 8           | UNIC_CUTFU_RC_BadUtf8Char | **Meaning:** A character in the source operand was not a valid UTF-8 character.  
**Action:** Make sure that the source operand contains only valid UTF-8 characters. |
| 1C          | UNIC_CUTFU_RC_WorkareaNotAligned | **Meaning:** The workarea provided was not on a doubleword boundary.  
**Action:** Make sure that the workarea is on a doubleword boundary. |

**Examples**

**Operation:**
Execute a MVCLU operation.

The code is as follows.

```
LA 2,MYPBLOCK           Get address of parm
USING UNIC_MVCLU,2
XC UNIC_MVCLU(UNIC_MVCLU_LEN),UNIC_MVCLU Clear block
* Also includes ALETs
MVC UNIC_MVCLU_TARGETADDR,TARGADDR Set target area
MVC UNIC_MVCLU_TARGETLEN,TARGLEN Set target length
MVC UNIC_MVCLU_SOURCEADDR,SOURCEADDR Set source area
MVC UNIC_MVCLU_SOURCELEN,SOURCELEN Set source length
MVC UNIC_MVCLU_PAD,PADCHAR Set pad char
LA 3,WORKAREA
ST 3,UNIC_MVCLU_WORKAREAADDR Set workarea address
CSRUNIC FUNCTION=MVCLU,PBLOCK=UNIC_MVCLU
DROP 2

DS 0F  Align parameter on word boundary
MYPBLOCK DS (UNIC_MVCLU_LEN)CL1 PBLOCK parameter
TARGADDR DS A  Output target area
TARGLEN DS F  Length of target area
SOURCEADDR DS A  Input source area
SOURCELEN DS F  Length of source area
PADCHAR DC XL2'4040' Pad with X'4040'
DS 0D  Doubleword align workarea
WORKAREA DS CL512  Work area
```
Chapter 81. CSRVIEW — View an object

Description

Call the CSRVIEW window service to:

- Map a window to one or more blocks of a data object. If you specified scrolling when you called CSRIDAC to identify the object, CSRVIEW maps the window to the blocks in the scroll area and maps the scroll area to the object.
- Specify that the reference pattern you are using is either random or sequential.
- End a view that you previously created through CSRVIEW or CSREVVW, and unmap the object.

Mapping a data object enables your program to access the data that is viewed through the window the same way it accesses other data in your storage.

The CSREVVW service also maps a data object. Use that service if your program references the data in the window in a sequential pattern and can benefit from having more than 16 blocks come into a window at one time, or if it can benefit from having fewer than 16 at one time.

Environment

The requirements for the caller are:

- Minimum authorization: Problem state with PSW key 8-15
- Dispatchable unit mode: Task
- Cross memory mode: PASN=HASN=SASN
- AMODE: 24- or 31-bit, but all addresses must be 31-bit addresses
- ASC mode: Primary
- Interrupt status: Enabled for I/O and external interrupts
- Locks: No locks held
- Control parameters: Must be in the primary address space.

Programming requirements

None.

Restrictions

The caller must follow all the restrictions imposed by the DIV macro.

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of a standard 18-word save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
CSRVIEW callable service

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

```
CALL CSRVIEW
   ,operation_type
   ,object_id
   ,offset
   ,span
   ,window_name
   ,usage
   ,disposition
   ,return_code
   ,reason_code
```

Parameters

All input to callable services is in the form of RX-type addresses.

The parameters are explained as follows:

- `(operation_type)`
  - Specifies the type of operation CSRVIEW is to perform. To begin viewing an object, specify BEGIN. To end a view, whether mapped by CSRVIEW or CSREVW, specify END.
  - Define `operation_type` as character data of length 5. If you specify END, pad the string on the right with 1 or 2 blanks.

- `(object_id)`
  - Specifies the object identifier. Supply the object identifier that CSRIDAC returned when you obtained access to the object.
  - Define `object_id` as character data of length 8.

- `(offset)`
  - Specifies the offset of the view into the object. Specify the offset in blocks of 4096 bytes.
  - Define `offset` as integer data of length 4.
Specifies the window size in blocks of 4096 bytes.
Define span as integer data of length 4.

Specifies the symbolic name you assigned to the window in your address space.

Specifies the expected pattern of references to pages in the object. Specify one of the following values:

**SEQ**
The reference pattern is expected to be sequential. If you specify SEQ, window services brings up to 16 blocks of data into the window at a time, depending on the size of the window and availability of resources.

**RANDOM**
The reference pattern is expected to be random. If you specify RANDOM, window services brings data into the window one block at a time.

Define usage as character data of length 6. If you specify SEQ, pad the string on the right with 1 to 3 blanks.

Defines how CSRVIEW is to handle data that is in the window when you begin or end a view.

- **When you specify CSRVIEW BEGIN and a disposition of:**
  
  **REPLACE**
  The first time you reference a block to which the window is mapped, CSRVIEW replaces the data in the window with the data from the referenced block.

  **RETAIN**
  When you reference a block to which the window is mapped, the data in the window remains unchanged. When you call CSRSAVE to save the mapped blocks, CSRSAVE saves all of the mapped blocks because CSRSAVE considers them changed.

- **When you specify CSRVIEW END and a disposition of:**
  
  **REPLACE**
  CSRVIEW discards the data that is in the window, making the window contents unpredictable. CSRVIEW does not update mapped blocks of the object or scroll area.

  **RETAIN**
  If the object is permanent and has no scroll area, CSRVIEW retains the data that is in the window. CSRVIEW does not update mapped blocks of the object.

  If the object is permanent and has a scroll area, or if the object is temporary, CSRVIEW retains the data that's in the window and updates the mapped blocks of the object or scroll area.

Define disposition as character data of length 7. If you specify RETAIN, pad the string on the right with a blank.

When CSRVIEW completes, return_code contains the return code. Define return_code as integer data of length 4.
CSRVIEW callable service

,reason_code
When CSRVIEW completes, reason_code contains the reason code. Define reason_code as integer data of length 4.

ABEND codes
The CSRVIEW service might abnormally terminate with abend code X'019'. See z/OS MVS System Codes for an explanation and programmer responses.

Return and reason codes
When the CSRVIEW service returns control to your program, GPR 15 (and return_code) contains a return code and GPR 0 (and reason_code) contains a reason code. The following table identifies return code and reason code combinations and tells what each means.

A return code of X'4' or X'C' means that data-in-virtual encountered a problem or an unexpected condition. Data-in-virtual reason codes, which are two bytes long and right justified, are explained in the description of the DIV macro (Chapter 88, "DIV — Data-in-virtual," on page 463).

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00000000                 | 00000000                | **Meaning**: The operation was successful.  
**Action**: None. |
| 00000004                 | 00000125                | **Meaning**: System error. The service could not retain all the data that was in the scroll area.  
**Action**: Retry the request. If the problem persists, contact the appropriate IBM support personnel. |
| 00000004                 | xxxxnnnn                | **Meaning**: The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.  
**Action**: See the DIV macro description for an explanation of nnnn. |
| 0000000C                 | xxxxnnnn                | **Meaning**: The value nnnn is a data-in-virtual reason code. The value xxxx is not part of the intended programming interface.  
**Action**: See the DIV macro description for an explanation of nnnn. |
| 0000002C                 | 00000004                | **Meaning**: Program error. Window services have not been defined to your system, or the link to the service failed.  
**Action**: If window services are available on your system, rerun the program one or more times. If the problem persists, contact the appropriate IBM support personnel. |
Chapter 82. CSVAPF — Query the list of APF-authorized libraries

Description

The CSVAPF macro allows you to determine the format and contents of the APF-authorized library list. You can issue CSVAPF to:

- Determine whether or not a library is in the APF list
- Determine the current format (dynamic or static) of the APF list
- Obtain a list of all library entries in the APF list.

You can issue CSVAPF to perform any of the listed functions on either a dynamic or static APF list.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: For a QUERY or QUERYFORMAT request, 31-bit. For a LIST request, 24- or 31-bit.
ASC mode: For a QUERY request, primary. For all other requests, primary or access register (AR).
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

If you code the LIST option on the REQUEST parameter, you must include the CSVAPF mapping macro (see z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/). For all other requests, you can optionally include the CSVAPF mapping macro to define variables and values for:

- Return and reason codes returned by CSVAPF
- The APF list format, which is returned by CSVAPF when you specify REQUEST=QUERYFORMAT.

Restrictions

None.

Input register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>For a QUERY request, the address of a standard 72-byte save area</td>
</tr>
</tbody>
</table>

Output register information

When control returns to the caller, the GPRs contain:
CSVAPF macro

Register Contents
0 If REQUEST=QUERYFORMAT is not specified, and the value in register 15 is not 0, reason code; otherwise, used as a work register by the system
1 Used as a work register by the system
2-13 Unchanged
14 Used as a work register by the system
15 For a QUERYFORMAT request, used as a work register by the system; for all other requests, return code

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

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

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

Performance implications
None.

Syntax
The standard form of the CSVAPF macro is written as follows:

name name: symbol. Begin name in column 1.

b One or more blanks must precede CSVAPF.

CSVAPF

b One or more blanks must follow CSVAPF.

Valid parameters (Required parameters are underlined):
REQUEST=QUERY
REQUEST=QUERYFORMAT
REQUEST=LIST
,DSNAME=libname
,VOLTYPE=SMS
,VOLTYPE=ANY,
,VOLUME=volume
,FORMAT=format
,ANSAREA=ansarea

libname: RS-type address or address in register (2) - (12).
Default: VOLTYPE=SMS
VOLUME is required with VOLTYPE=ANY.
volume: RS-type or address in register (2) - (12).
format: RS-type address or address in register (2) - (12).
ansarea: RS-type address or address in register (2) - (12).
### Parameters

The parameters are explained as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST=QUERY</td>
<td>Determine if a particular library is in the APF list.</td>
</tr>
<tr>
<td>REQUEST=QUERYFORMAT</td>
<td>Determine the current format (dynamic or static) of the APF list. The system returns information to the one byte field specified on the FORMAT parameter. If the output is 00, the list is static; if the output is 01, the list is dynamic. When you specify this parameter, you cannot specify the RETCODE, RSNCODE, and MF parameters. The system does not provide return and reason codes for a QUERYFORMAT request.</td>
</tr>
<tr>
<td>REQUEST=LIST</td>
<td>Request a list of the libraries in the APF list. The system returns the list to the area specified by the ANSAREA parameter. See the description of the ANSAREA parameter for information on how to read the entries in the list.</td>
</tr>
</tbody>
</table>

**Note:** The list will include those libraries that are defined or defaulted to be APF-authorized. The definition could be via IEAAPFx or PROGxx parmlib members, the CSVAPF macro, or the SETPROG APF system command. Note that programs that are marked as coming from an authorized library could have come from one of these libraries or from the link pack area.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNAME=libname</td>
<td>Specifies a field (or a register containing the address of a field) containing a 44-character name of an APF-authorized library. If the library name is less than 44 characters, it must be left-justified in a 44-character field and padded with blanks.</td>
</tr>
</tbody>
</table>
CSVAPF macro

You can specify an alias of an APF-authorized library instead of the actual library name. However, the CSVAPF service considers an alias to be APF-authorized only when it is defined in the APF list.

**Note:** Usually, you do not need to define the alias of an APF-authorized library in the APF list. IBM's data management services (for example, OPEN processing) map an alias to the actual library name, and therefore does not require the alias to be defined in the APF list. An alias must be defined in the APF list only when the alias is to be used as input to the CSVAPF QUERY macro request, or on the SETPROG APF or DISPLAY PROG,APF operator commands.

',VOLTYPE=SMS
',VOLTYPE=ANY, VOLUME=volume

Specifies the status of the library specified on the DSNAME parameter, which is one of the following:

**SMS** The library is managed by the storage management subsystem (SMS).

**ANY** The library may or may not be SMS-managed. The library is located on volume volume, which specifies the address of a 6-character volume serial number; for an ADD request, you can also specify ****** (six asterisks) to indicate the current sysres volume, or "MCAT" to indicate the volume on which the master catalog resides. If volume is all zeros, the system assumes that the library is SMS-managed.

**Note:** The return code on a Query is determined by whether the match is exact or inexact.

A return code of 0 indicates an exact match which could be:
- You coded DSNAME=d and VOLTYPE=ANY and VOLUME=v and there is an entry in the APF list that matches both the data set and the volser.
- You coded DSNAME=d and an indication of “SMS-managed” (VOLTYPE=SMS) and there is an entry in the APF list that matches the data set and indicates “SMS-managed”.

A return code of 4 with a reason code = 0401 indicates an inexact match which is:
- You coded DSNAME=d and VOLTYPE=ANY and VOLUME=v and there is no exact match, but there is an entry in the APF list that matches the data set and indicates “SMS-managed”.

',FORMAT=format

Specifies a 1-byte field (or a register containing the address of a field) for output that the system is to use to indicate the current format of the APF list.

',ANSAREA=ansarea

Specifies an area (or a register containing the address of an area) where the system is to store the current list of APF-authorized libraries. Use the CSVAPFAA mapping macro to map this area. Specify the length of this area on the ANSLEN parameter.

The system returns a header that indicates the total number of libraries in the list and the offset to the first library entry. To find the next entry, add the value in the length field (APFELEN) to the address of the current entry.
CSVAPF macro

For each library entry, the volume identifier in field APFEVOLUME is valid only when the library is not SMS-managed (the bit APFESMS in field APFEFLAGS is off). If the library is SMS-managed, field APFEVOLUME contains "*SMS* ".

\texttt{\textbackslash ANSLEN=anslen}

Specifies a fullword (or a register containing the address of a fullword) that contains the length of the area where the system is to return the current APF list. This value must be equal to or greater than the length of the APFHDR structure in the CSVAPFAA mapping macro.

If the area is not long enough to contain the entire APF list, the system returns as many entries as it can provide. The system indicates the length that is currently required to contain all the information in field APFHTLEN in the CSVAPFAA mapping macro.

\texttt{\textbackslash RETCODE=retcode}

Specifies a fullword (or a register) where the system is to store the return code. The return code is also in general purpose register (GPR) 15. Do not specify this parameter on a QUERYFORMAT request.

\texttt{\textbackslash RSNCODE=rsncode}

Specifies a fullword (or a register) where the system is to store the reason code. The reason code is also in general purpose register (GPR) 0. Do not specify this parameter on a QUERYFORMAT request.

\texttt{\textbackslash MF=S}

Specifies the standard form of the CSVAPF macro. Do not specify this parameter on a QUERYFORMAT request.

**ABEND codes**

None.

**Return and reason codes**

When the CSVAPF macro returns control to your program, GPR 15 (and retcode) contains a return code. When the value in GPR 15 is not zero, GPR 0 (and rsncode) contains a reason code. xxxx indicates internal information. If you specified the QUERYFORMAT option, CSVAPF does not return any return or reason code to your program.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | —                       | **Meaning**: The CSVAPF request completed successfully. The result depends on the option:  
|                         |                         | • QUERY - The system found the library in the APF list.  
|                         |                         | • LIST - The system returned a list of all the libraries in the APF list.  
|                         |                         | **Action**: None. |
| 04                      | xxxx0401                | **Meaning**: For a QUERY request, the library is in the list, and is SMS-managed.  
<p>|                         |                         | <strong>Action</strong>: None. |</p>
<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04                      | xxxx0402                | **Meaning**: For a QUERY request, the library is not in the APF list.  
**Action**: None. |
| 04                      | xxxx0403                | **Meaning**: Program error. For a LIST request, the value specified on the ANSLEN parameter is not large enough to contain the entire list of APF-authorized libraries.  
**Action**: Check the answer area field APFHTLEN in the CSVAPFAA mapping macro to see how much space is required to return the APF list. Issue the CSVAPF macro again, specifying, on the ANSLEN parameter, a fullword containing a value large enough to contain the entire APF list. |
| 08                      | xxxx0801                | **Meaning**: Program error. The system could not access the parameter list that the CSVAPFAA macro created.  
**Action**: Ensure that the parameter list is addressable. |
| 08                      | xxxx0804                | **Meaning**: Program error. The caller is not authorized to issue the CSVAPF macro for the specified request.  
**Action**: See the authorization requirements described in the Environment section for this macro. |
| 08                      | xxxx0805                | **Meaning**: Program error. The system could not perform the function because the home address space is different from the primary address space.  
**Action**: For the specified request, do not issue the CSVAPF macro while running in cross memory mode. |
| 08                      | xxxx0806                | **Meaning**: Program error. The ALET of the area specified on the ANSAREA parameter is not correct.  
**Action**: Ensure that the ALET is 0, or that the ALET represents a valid entry on the DU-AL. If you specified register notation "(n)," make sure that the ALET in register n is correct. |
| 08                      | xxxx0807                | **Meaning**: Program error. The system found an error when accessing the answer area specified on the ANSAREA parameter.  
**Action**: Ensure that the answer area address specified on the ANSAREA parameter is valid. |
### CSVAPF macro

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 08 xxxx0808             |                         | **Meaning:** Program error. For a QUERY request, the length of the answer area specified on the ANSLEN parameter is not equal to or greater than the length of the APFHDR structure in the CSVAPFAA mapping macro.  
**Action:** On the ANSLEN parameter, specify a fullword containing a value that is equal to or greater than the length of the APFHDR structure in the CSVAPFAA mapping macro. |
| 08 xxxx0809             |                         | **Meaning:** Program error. The request type is not valid.  
**Action:** Check for a possible overlay in the parameter list that the CSVAPFAA mapping macro created. |
| 08 xxxx080A             |                         | **Meaning:** Program error. The CSVAPF macro could not establish an ESTAEX recovery routine. xxxx is the return code from the ESTAEX service.  
**Action:** See the description of the ESTAEX macro for the action associated with the xxxx return code. |
| 08 xxxx080B             |                         | **Meaning:** Program error. A reserved field is not zero in the parameter list that the CSVAPFAA macro created.  
**Action:** Check for a possible overlay in the parameter list that the CSVAPFAA macro created. |
| 08 xxxx080C             |                         | **Meaning:** Program error. The library name specified on the DSNAME parameter is not valid. The first character is blank.  
**Action:** On the DSNAME parameter, specify a library name that does not include a blank as the first character. |
| 08 xxxx080D             |                         | **Meaning:** Program error: The system found an error in the access list entry token (ALET) for the parameter list that the CSVAPFAA macro created.  
**Action:** Ensure that the ALET is 0 or that the ALET represents a valid entry on the DU-AL. |
| 08 xxxx080E             |                         | **Meaning:** Program error. The system found an incorrect version number in the parameter list that the CSVAPF macro created.  
**Action:** Verify that your program is not overwriting the parameter list, and that the execute form of the macro correctly addresses the parameter list. If you are using the modify form of the macro, make sure that you specified the COMPLETE option on at least one invocation. |
### CSVAPF macro

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>xxxx1001</td>
<td><strong>Meaning</strong>: System error. An internal error occurred. <strong>Action</strong>: Contact the system programmer. Provide the return code, the reason code, and the explanation of the error.</td>
</tr>
</tbody>
</table>

**Example 1**

Determine the current format of the APF list:

```
CSVAPF REQUEST=QUERYFORMAT,FORMAT=LFORMAT
CLI LFORMAT,CSVAPFFORMATDYNAMIC
BE LAB1
```

```
* Format is static
.
LAB1 DS 0H Format is dynamic
.
```

```
LFORMAT DS X Output Format
CSVAPFAA , Include CSVAPFAA mapping
```

**Example 2**

Change a program to use the CSVAPF macro to access the APF list (this program uses the LIST function as an example of one way to access the APF list):

```
L 15,X'10' Get CVT address
TM CVTDCB-CVTMAP(15),CVTOSEXT OS Extension present
BZ OLDLIST No, old (static) list
TM CVOTOSLV1-CVTMAP(15),CVDYAPF Is dynamic APF present?
BZ OLDLIST No, old (static) list
MVC APAALEN,-AL4(4096) Assume length is 4K
L 2,APAALEN Get length
GETMAIN RU,LV=(2) Get storage for answer area
ST 1,APAAL0 Save answer area address
LAB1 DS 0H
```

```
L 4,APAAL0 Get answer area address
CSVAPF REQUEST=LIST,ANSAREA=(4),ANSLEN=APAALEN, RETCODE=RETCODE,RSNCODE=RSNICODE
CLC RETCODE,=AL4(CSVAPFRC_OK) Success?
BE LAB3 Yes, process data
CLC RETCODE,=AL4(CSVAPFRC_WARN) Warning?
BNE LAB2 No, Process other return codes
NC RSNICODE,=AL4(CSVAPFRSNCODEMASK) Clear high order bits
CLC RSNICODE,=AL4(CSVAPFRSNOTALLDATARETURNED) More data?
BNE LAB2 No, Process other return codes
L 3,APAALEN Get current length
L 2,APFHTLEN-APFHDR(4) Get required length
ST 2,APAALEN Save total required length
FREEMAIN RU,LV=(3),A=(4) Free previous area
GETMAIN RU,LV=(2) Get storage for answer area
ST 1,APAAL0 Save answer area address
B LAB1 Re-do LIST request
LAB2 DS 0H Process other return codes
```

* Current code to process static format APF list
**CSVAPF—List form**

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

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

```assembly
name name: symbol. Begin name in column 1.

b One or more blanks must precede CSVAPF.

CSVAPF

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

**Parameters**

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

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

---

`APAA@ DS A Address of APF answer area
APAALEN DS F Length of APF answer area
LRETCODE DS F Return code
LRSNCCODE DS F Reason code
CSVAPFAA , Include CSVAPFAA mapping`
CSVAPF macro

`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.

CSVAPF—Execute form

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

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

```
  name  name: symbol. Begin name in column 1.
b
CSVPF
b
```

Valid parameters (Required parameters are underlined):

<table>
<thead>
<tr>
<th>REQUEST=QUERY</th>
<th>REQUEST=LIST</th>
<th>,DSNAME=dsname</th>
<th>,VOLTYPE=SMS</th>
<th>,VOLTYPE=ANY, VOLUME=volume</th>
<th>,FORMAT=format</th>
<th>,ANSAREA=ansarea</th>
<th>,ANSLEN=anslen</th>
<th>,RETCODE=retcode</th>
<th>,RSNCODE=rsncode</th>
<th>,MF=(E, list addr)</th>
<th>,MF=(E, list addr, COMPLETE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNAMES, VOLTYPE, VOLUME, RETCODE, RSNCODE</td>
<td>ANSAREA, ANSLEN, RETCODE, RSNCODE</td>
<td>dsname: RS-type address or register (2) - (12).</td>
<td>Default: VOLTYPE=SMS</td>
<td>VOLUME is required with VOLTYPE=ANY.</td>
<td>volume: RS-type or register (2) - (12).</td>
<td>format: RS-type address, or register (2) - (12).</td>
<td>ansarea: A-type address, or register (2) - (12).</td>
<td>anslen: A-type address, or register (2) - (12).</td>
<td>retcode: RS-type address or register (2) - (12).</td>
<td>rsncode: RS-type address or register (2) - (12).</td>
<td>list addr: RS-type address, or register (2) - (12).</td>
</tr>
</tbody>
</table>
Parameters

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

\[ \text{MF} = (E, \text{list addr}) \]
\[ \text{MF} = (E, \text{list addr}, \text{COMPLETE}) \]

Specifies the execute form of the CSVAPF macro.

\text{list addr} specifies the area that the system uses to store the parameters.

\text{COMPLETE}, which is the default, specifies that the system is to check for required parameters and supply optional parameters that you did not specify.
CSVAPF macro
Chapter 83. CSVINFO — Obtain information about loaded modules

Description

Use CSVINFO to obtain information about modules:
- Loaded into the link pack area (LPA): specify FUNC=LPA
- Loaded into the job pack area (JPA): specify FUNC=JPA
- Loaded by a specific task using the LOAD macro: specify FUNC=TASKLOAD
- Running under all program request blocks (PRBs) and supervisor request blocks (SVRBs) associated with a specific task, including those that received control through the LINK(X), ATTACH(X), or XCTL(X) macro; or through the z/OS UNIX System Services EXEC command: specify FUNC=TASKALL
- Running under a specific PRB or SVRB: specify FUNC=RB
- Copied from the parent address space into the job pack area under the z/OS UNIX System Services fork process: specify FUNC=JPA.

When providing information about a loaded module, CSVINFO returns information separately for each of the following types of entry points:
- The major entry point
- Each entry point created using the IDENTIFY macro
- Each minor entry point specified on a LOAD, LINK(X), ATTACH(X), or XCTL(X) invocation the system is processing while CSVINFO is running
- The z/OS UNIX System Services entry point (including its file name), if the loaded module is an z/OS UNIX System Services module.

The CSVINFO macro can return information about one loaded module (such as the module running under a specific PRB) or group of loaded modules (such as all modules in LPA). The CSVQUERY macro, which also provides information about loaded modules, returns information about only one particular loaded module at a time.

CSVINFO obtains information about one loaded module at a time, stores the information in the CSVMODI data area, and passes the data area to a user-written module information processing routine (MIPR). The MIPR examines this data and returns control to CSVINFO, either requesting information about an additional loaded module or indicating that no more information is needed. For instance, if you request information for all modules loaded by a particular task, CSVINFO calls the MIPR multiple times, passing information about each loaded module of interest. CSVINFO continues to pass loaded module information to the MIPR until either of the following occurs:
- CSVINFO has returned all available information.
- The MIPR indicates that no more information is needed by returning a nonzero return code to CSVINFO.

You can issue the CSVINFO macro from a program to obtain information about loaded modules in system storage, or from an IPCS exit to search a dump for information about loaded modules.
CSVINFO macro

References
For detailed information about any of the following, see the program management topic in z/OS MVS Programming: Assembler Services Guide:
- How the CSVINFO macro compares with the CSVQUERY macro
- How to use the CSVINFO macro
- How to code a MIPR
- Load modules and their characteristics

Typically, a pathname returned from CSVINFO is prefixed by a slash (/); however, if that pathname was returned in response to a load (BPX1LOD), exec (BPX1EXC) or spawn (BPX1SPN) call where the HFS program was found in the current working directory, the pathname will not be prefixed with a "/". To determine the full pathname of the HFS program in this case, call BPX1GCW to obtain the current working directory name that was used to locate the program. You can then use this directory pathname as returned from BPX1CGW to prefix the pathname returned by CSVINFO to determine the full pathname of the HFS program.

For information about the CSVMODI data area, see z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Environment

Requirements for the caller:

Minimum authorization: Problem state with any PSW key

Dispatchable unit mode: Task or SRB
Cross memory mode: PASN=HASN=SASN
AMODE: 31-bit
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: Supervisor state and PSW key 0 callers may hold the LOCAL and the CMS locks.

Control parameters: Must be in the primary address space

For additional information under Programming requirements.

Programming requirements

If you are requesting information about loaded modules in common storage or if multi-tasking is taking place in your address space, the module information you request might be changing while the CSVINFO service is retrieving information unless serialization has been obtained.

If your program runs in supervisor state and invokes the CSVINFO macro, the CSVINFO service obtains the appropriate locks if your program does not already hold them.

Other callers might receive incorrect data or end abnormally if the CSVINFO service accesses a data area that is being updated.
Restrictions

The TCB specified with the TCBADDR keyword must reside in the caller's primary address space unless the CSVINFO macro is being issued from an IPCS exit.

When you issue the CSVINFO macro from an IPCS exit, CSVINFO does not:
• Provide serialization
• Establish a recovery environment before passing control to your MIPR.

Input register information

Before issuing the CSVINFO 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

If you require information about a specific loaded module, use the CSVQUERY macro to obtain better performance.

Syntax

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

```
name name: Symbol. Begin name in column 1.
b One or more blanks must precede CSVINFO.
CSVINFO
b One or more blanks must follow CSVINFO.
```
CSVINFO macro

FUNC=LPA
FUNC=JPA,TCBADDR=tcbaddr
FUNC=TASKLOAD,TCBADDR=tcbaddr
FUNC=TASKALL,TCBADDR=tcbaddr
FUNC=RB,RBADDR=rbaddr
tcbaddr: RS-type address or address in register (2) - (12).
rbaddr: RS-type address or address in register (2) - (12).

,ENV=MVS
,ENV=IPCS,ABDPLPTR=abdplptr,ASID=asid
abdplptr: RS-type address or address in register (2) - (12).
asid: RS-type address or address in register (2) - (12).

,MIPR=mipr
mipr: RS-type address or address in register (2) - (12).

,USERDATA=userdata
userdata: RS-type address.

,COM=com
com: Comment text enclosed in single quotation marks.

,RETCODE=retcode
retcode: RS-type address or address in register (2) - (12).

,RSNCODE=rsncode
rsncode: RS-type address or address in register (2) - (12).

Parameters

The parameters are explained as follows:

FUNC=LPA
FUNC=JPA,TCBADDR=tcbaddr
FUNC=TASKLOAD,TCBADDR=tcbaddr
FUNC=TASKALL,TCBADDR=tcbaddr
FUNC=RB,RBADDR=rbaddr
A required parameter that specifies the function CSVINFO is to perform.

FUNC=LPA requests that CSVINFO place into the CSVMODI data area information about link pack area (LPA) modules. The search order for LPA modules is the active link pack area (MLPA and FLPA), followed by PLPA. If CSVINFO encounters more than one copy of a loaded module, CSVINFO provides information about each copy.

FUNC=JPA,TCBADDR=tcbaddr requests that CSVINFO place into the CSVMODI data area information for modules in the job pack area for the job step task TCB specified by tcbaddr. When you specify FUNC=JPA, CSVINFO retrieves information for:
- All modules in the private area known to the specified job step task
- All modules in common storage that have been loaded by an authorized task running under the specified job step task, using the LOAD macro with the GLOBAL parameter.

FUNC=TASKLOAD,TCBADDR=tcbaddr requests that CSVINFO place into the CSVMODI data area information about all modules loaded by the task specified by tcbaddr, using the LOAD macro. Only modules that have not yet been deleted are processed.

FUNC=TASKALL,TCBADDR=tcbaddr requests that CSVINFO place into the CSVMODI data area information about all modules running under PRBs and SVRBs under the task specified by tcbaddr, including all modules that have
CSVINFO macro

received control through the LINK(X), ATTACH(X), or XCTL(X) macro. FUNC=TASKALL returns information on LPA modules as well as private modules. If CSVINFO encounters more than one copy of a loaded module, CSVINFO provides information about each copy.

TCBADDR=tcbaddr specifies the address of a required 4-byte field that contains the address of the TCB about which you want information.

FUNC=RB, RBADDR=rbaddr requests that CSVINFO place into the CSVMODI data area information about the module running under the PRB or SVRB specified by rbaddr.

RBADDR=rbaddr specifies the address of a required 4-byte field that contains the address of the PRB or SVRB about which you want information.

,ENV=MVS
,ENV=IPCS,ABDPLPTR=abdplptr,ASID=asid
A required parameter that specifies whether you are issuing CSVINFO from a program (to search system storage) or from an IPCS exit (to examine a dump). ENV=MVS specifies that you are issuing CSVINFO from a program and that you want CSVINFO to examine system storage.

ENV=IPCS specifies that you are issuing CSVINFO from an IPCS exit to search a dump. When you specify ENV=IPCS you must also specify ABDPLPTR=abdplptr and ASID=asid.

ABDPLPTR=abdplptr specifies the address of the ABDUMP parameter list (ABDPL) that is currently in use. When your IPCS exit routine gets control, GPR 1 contains the address of the ABDUMP parameter list. CSVINFO passes the address of the ABDPL to the caller's MIPR in the input parameter list mapped by the CSVMODI mapping macro.

ASID=asid identifies the address space id (ASID) in the dump from which the requested module information is to be obtained. asid contains the address of a 16-bit address space identifier. The specified address space identifier is stored in the ADPLASID field of the ABDPL, and the ADPLASID field contains this value when CSVINFO passes control to your MIPR.

,MIPR=mipr
A required parameter that contains the address of the caller's module information processing routine (MIPR).

,USERDATA=userdata
Specifies the address of an optional 16-byte input field that contains user data to be passed to the MIPR. The CSVINFO macro places the user data into the CSVMODI data area before it passes control to the MIPR.

,COM=com
Specifies an optional character input. You can use this keyword to produce a comment in the macro expansion. The comment string must be enclosed in single quotation marks if it contains lowercase characters.

,RETCODE=retcode
Specifies the location where the system is to store the return code. The return code is also in GPR 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=rsncode
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0. If you specify a storage location, it must be on a fullword boundary.
CSVINFO macro

ABEND codes

None.

Return and reason codes

When CSVINFO returns control to your program, GPR 15 (and retcode, if you coded RETCODE) contains the return code.

For a return code of X'8', GPR 0 (and rsncode, if you coded RSNCODE) contains a reason code set by the MIPR. For other return codes, the reason code is always 0.

Table 13. Return Codes for the CSVINFO Macro

<table>
<thead>
<tr>
<th>Hexadecimal</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
<tr>
<td>4</td>
<td>Meaning: Successful completion. Action: None. There was no information for CSVINFO to return.</td>
</tr>
<tr>
<td>8</td>
<td>Meaning: CSVINFO processing was ended by a nonzero return code from the caller's MIPR. GPR 0 (and rsncode, if you coded RSNCODE) contains a reason code from the MIPR. Action: Check the reason code from the MIPR and take appropriate action.</td>
</tr>
<tr>
<td>C</td>
<td>Meaning: Program error. CSVINFO was unable to obtain the local lock needed for serialization for a supervisor state caller. Action: Release the CML lock before invoking CSVINFO.</td>
</tr>
<tr>
<td>10</td>
<td>Meaning: Program error. A parameter specified for CSVINFO was inaccessible or not valid. Action: Correct the parameters and rerun the program.</td>
</tr>
<tr>
<td>14</td>
<td>Meaning: Environmental error. The CSVINFO service should have been available but wasn't. Action: Ask the system programmer to determine why the CSVINFO service is unavailable.</td>
</tr>
<tr>
<td>18</td>
<td>Meaning: System or program error. CSVINFO processing ended because the requested information could not be retrieved from the dump. This return code applies only when CSVINFO is issued from an IPCS exit. The message BLS18100I accompanies this return code. See z/OS MVS Dump Output Messages for further information about this message. Action: Ensure that you have not passed the CSVINFO service an incorrect address and rerun the program. If the program receives this return code again, either the necessary data areas are not in the dump or there might be an error in the control blocks used to keep track of loaded modules.</td>
</tr>
<tr>
<td>1C</td>
<td>Meaning: System error. This return code is for IBM diagnostic purposes only. Action: Rerun the program one or more times. If the problem persists, record the return code and message text and supply it to the appropriate IBM support personnel.</td>
</tr>
<tr>
<td>20</td>
<td>Meaning: Environmental error. The CSVINFO service is not supported on this level of the system. Action: Check with your system programmer to determine which system your program should run on to use the CSVINFO service.</td>
</tr>
</tbody>
</table>
### CSVINFO macro

#### Table 13. Return Codes for the CSVINFO Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 24                      | **Meaning:** Environmental error. The CSVINFO parameter list is not valid with the level of CSVINFO service on the system.  
**Action:** Record the return code and supply it to the appropriate IBM support personnel. |
| 28                      | **Meaning:** System error. CSVINFO timed out after entering an infinite loop while accessing information about loaded modules.  
**Action:** If you specified ENV=MVS and your program was not in supervisor state, rerun the program. The error might have been temporary, resulting from a lack of serialization while accessing control blocks.  
If the error persists or CSVINFO was running with serialization when the error occurred, record the return code and supply it to the appropriate IBM support personnel. |
| 2C                      | **Meaning:** Program error. The RB address specified using the RBADDR parameter on a FUNC=RB request is not the address of a PRB or an SVRB.  
**Action:** Ensure that you pass the address of a PRB or an SVRB. CSVINFO does not process requests for other types of RBs. |
| 30                      | **Meaning:** Program error. The MIPR failed.  
**Action:** Ensure that the MIPR restores GPRs 2-13 before returning control to CSVINFO. If this was not the problem and the MIPR did not have its own recovery routine, your options depend on whether your program was running in an authorized state.  
CSVINFO’s recovery routine issued an SVC dump if your program was authorized in at least one of the following ways:  
• Supervisor state  
• PSW key 0-7  
• APF authorization.  
If a dump was taken, examine it for information about why the MIPR might have failed.  
If your program was running in an unauthorized state, the information recorded in the job log at the time of the failure is the only information provided. |
| 34                      | **Meaning:** System error. While processing the RB chain, CSVINFO entered an infinite loop, signalled by reaching 1000 iterations. The RB address specified on the RBADDR parameter for a FUNC=RB request caused a circular RB chain.  
**Action:** Ensure that you pass the address of a PRB or an SVRB. CSVINFO does not process requests for other types of RBs.  
If you passed a valid RB address, record the return code and supply it to the appropriate IBM support personnel. |

### CSVINFO-List form

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

name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede CSVINFO.

CSVINFO

b

One or more blanks must follow CSVINFO.

<table>
<thead>
<tr>
<th>MF=(L,list addr)</th>
<th>list addr: RS-type address or register (2) - (12).</th>
</tr>
</thead>
<tbody>
<tr>
<td>MF=(L,list addr, attr)</td>
<td>attr: 1- to 60-character string input.</td>
</tr>
<tr>
<td>MF=(L,list addr, 0D)</td>
<td>Default: 0D.</td>
</tr>
</tbody>
</table>

Parameters

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

- MF=(L,list addr)
- MF=(L,list addr, attr)
- MF=(L,list addr, 0D)

Specifies the list form of the CSVINFO 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.

CSVINFO-Execute form

Use the execute form of the CSVINFO 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 CSVINFO macro is written as follows:
### CSVINFO macro

- `FUNC=LPA`
- `FUNC=JPA,TCBADDR=tcbaddr`
- `FUNC=TASKLOAD,TCBADDR=tcbaddr`
- `FUNC=TASKALL,TCBADDR=tcbaddr`
- `FUNC=RB,RBADDR=rbaddr`

  * `tcbaddr`: RS-type address or address in register (2) - (12).
  * `rbaddr`: RS-type address or address in register (2) - (12).

- `.ENV=MVS`
- `.ENV=IPCS,ABDPLPTR=abdppt;ASID=asid`

  * `abdppt`: RS-type address or address in register (2) - (12).
  * `asid`: RS-type address or address in register (2) - (12).

- `.MIPR=mipr`

  * `mipr`: RS-type address or address in register (2) - (12).

- `.USERDATA=userdata`
  - `userdata`: RS-type address.
  - **Default**: NULL

- `.COM=com`
  - `com`: Comment text enclosed in single quotation marks.
  - **Default**: NULL

- `.RETCODE=retcode`

  * `retcode`: RS-type address or address in register (2) - (12).

- `.RSNCODE=rsncode`

  * `rsncode`: RS-type address or address in register (2) - (12).

- `.MF=(E, list addr)`
  - **Default**: COMPLETE

### Parameters

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

- `.MF=(E, list addr)`
- `.MF=(E, list addr, COMPLETE)`
- `.MF=(E, list addr, NOCHECK)`

  Specifies the execute form of the CSVINFO 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 defaults for omitted optional parameters.

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

### CSVINFO—Modify form

Use the modify form of the CSVINFO 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 the storage area; use the modify form to set the appropriate options; then use the execute form to call the service.
The modify form of the CSVINFO macro is written as follows:

```
name
```

name: Symbol. Begin name in column 1.

b

One or more blanks must precede CSVINFO.

CSVINFO

b

One or more blanks must follow CSVINFO.

```
FUNC=LPA
FUNC=JPA,TCBADDR=tcbaddr
FUNC=TASKLOAD,TCBADDR=tcbaddr
FUNC=TASKALL,TCBADDR=tcbaddr
FUNC=RB,RBADDR=rbaddr
tcbaddr: RS-type address or address in register (2) - (12).
rbaddr: RS-type address or address in register (2) - (12).

,ENV=MVS
,ENV=IPCS,ABDPLPTR=abdpiptr,ASID=asid

abdpiptr: RS-type address or address in register (2) - (12).
asid: RS-type address or address in register (2) - (12).

,MIPR=mipr

mipr: RS-type address or address in register (2) - (12).

,USERDATA=userdata
,USERDATA=NULL

userdata: RS-type address.  
Default: NULL

,COM=com
,COM=NULL

com: Comment text enclosed in single quotation marks.  
Default: NULL

,RETCODE=retcode

retcode: RS-type address or address in register (2) - (12).

,RSNCODE=rsncole

rsncole: RS-type address or address in register (2) - (12).

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

list addr: RS-type address or register (2) - (12).

Default: COMPLETE
```

Parameters

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

```
,MF=(M,list addr)
,MF=(M,list addr,COMPLETE)
```
CSVINFO macro

,MF=(M,list addr,NOCHECK)
Specifies the modify form of the CSVINFO 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 defaults for omitted optional parameters.

NOCHECK specifies that the system is not to check for required parameters
and is not to supply defaults for omitted optional parameters.
CSVINFO macro
Chapter 84. CSVQUERY — Contents supervisor query service

Description

Use CSVQUERY to obtain information about the attributes of a loaded module residing in the job pack area (JPA) of the current primary address space or the link pack area (LPA). Specify the module you want information about, using an entry point name, entry point token, or any address within the loaded module. See the INEPTKN parameter description for information about obtaining an entry point token.

CSVQUERY returns information for the following types of entry points:
- Major entry points
- Entry points created using the IDENTIFY macro
- Minor entry points specified on a LOAD, LINK(X), ATTACH(X), or XCTL(X) invocation the system is processing while CSVQUERY is running.

For information about load modules and their characteristics, as well as a comparison of the CSVQUERY and CSVINFO macros, see the program management topic in z/OS MVS Programming: Assembler Services Guide.

Environment

Requirements for CSVQUERY callers are:

Minimum authorization: Problem state and any PSW key
Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any SASN, any HASN
AMODE: 24 or 31- or 64-bit
ASC mode: Primary or access register (AR)
Interrupt status: Enabled for I/O and external interrupts
Locks: The caller may hold the LOCAL lock of the current primary address space (if the home address space is the same as the current primary address space, this is the LOCAL lock) and may hold the CMS lock, but is not required to hold any locks.
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).

Input register information

Before issuing the CSVQUERY 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>
CSVQUERY macro

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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.

Programming requirements

If the program is in AR mode, issue the SYSSTATE macro with the ASCENV=AR parameter before you issue CSVQUERY. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

Restrictions

None.

Performance implications

If you specify an address as a search argument for a module in the PLPA, the search might take longer than if you specify a name because the PLPA is organized by name. You can obtain the best performance on a CSVQUERY request by specifying an entry point token.

Syntax

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

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

One or more blanks must precede CSVQUERY.

CSVQUERY

One or more blanks must follow CSVQUERY.
```

```
INENNAME=entryname entryname: RS-type address or register (2) - (12).
INEPTKN=ineptkn ineptkn: RS-type address or register (2) - (12).
INADDR=ptr name ptr name: RS-type address or register (2) - (12).
INADDR64=ptr name64 ptr name64: RS-type address or register (2) - (12).

,SEARCH=JPALPA Default: JPALPA
,SEARCH=JPA
,SEARCH=LPA

,SEARCHMINOR=NO Default: NO
,SEARCHMINOR=YES
```
CSVQUERY macro

.OUTLENGTH=length
  
  length: RS-type address or register (2) - (12).

.OUTLENGTH64=length64
  
  length64: RS-type address or register (2) - (12).

.OUTEPNM=entrename
  
  entrename: RS-type address or register (2) - (12).

.OUTEPTKN=outeptkn
  
  outeptkn: RS-type address or register (2) - (12).

.OUTEPA=entry addr
  
  entry addr: RS-type address or register (2) - (12).

.OUTEPA64=entry addr64
  
  entry addr64: RS-type address or register (2) - (12).

.OUTMJNM=major name
  
  major name: RS-type address or register (2) - (12).

.OUTLOADPT=outloadpt
  
  outloadpt: RS-type address or register (2) - (12).

.OUTLOADPT64=outloadpt64
  
  outloadpt64: RS-type address or register (2) - (12).

.OUTSP=subpool
  
  subpool: RS-type address or register (2) - (12).

.OUTATTR1=attr1
  
  attr1: RS-type address or register (2) - (12).

.OUTATTR2=attr2
  
  attr2: RS-type address or register (2) - (12).

.OUTATTR3=attr3
  
  attr3: RS-type address or register (2) - (12).

.OUTDIAG=outdiag
  
  outdiag: RS-type address or register (2) - (12).

.OUTRTID=outrtid
  
  outrtid: RS-type address or register (2) - (12).

.OUTXATTR1=xattr
  
  xattr: RS-type address or register (2) - (12).

.OUTVALID=valid
  
  valid: RS-type address or register (2) - (12).

.OUTPDATA=outpdata
  
  outpdata: RS-type address or register (2) - (12).

.OUTPID=outpid
  
  outpid: RS-type address or register (2) - (12).

.OUTXTLST=xtlst
  
  xtlst: RS-type address or register (2) - (12).

.OUTXTLST64=xtlst64
  
  xtlst64: RS-type address or register (2) - (12).

.OUTPATHNAME=pathname
  
  pathname: RS-type address or register (2) - (12).

.OUTDSKEY=outskey
  
  outskey: RS-type address or register (2) - (12).

,PLISTVER=IMPLIED_VERSION
  
  Default: IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=plistver

.RETCODE=retcode
  
  retcode: RS-type address or register (2) - (12).

,MF=S
Parameters

The parameters are explained as follows:

\texttt{INEPNAME=entryname}
\texttt{INEPTKN=ineptkn}
\texttt{INADDR=ptr name}
\texttt{INADDR64=ptr name64}

\texttt{INEPNAME=entryname} specifies an 8-character variable that contains the name of the entry point. The entry point name must be eight characters long, padded to the right with blanks if necessary.

\texttt{INEPTKN=ineptkn} specifies an 8-character variable that contains the entry point token. An entry point token is a unique, 8-character token assigned to each loaded module. To obtain the input token, invoke the CSVQUERY macro with \texttt{INADDR}, \texttt{INADDR64}, or \texttt{INEPNAME}, specifying the \texttt{OUTEPTKN} parameter. Use the output entry point token from that invocation of CSVQUERY as the input entry point token on subsequent invocations of CSVQUERY for the same module.

\texttt{INADDR=ptr name} specifies an address that CSVQUERY attempts to match to a loaded module. The address may be anywhere within the module.

\texttt{INADDR64=ptr name64} specifies an 8-byte address that CSVQUERY attempts to match to a loaded module. The address may be anywhere within the module.

You must specify one of the following mutually exclusive parameters: \texttt{INEPNAME}, \texttt{INEPTKN}, \texttt{INADDR}, or \texttt{INADDR64}.

\texttt{,SEARCH=JPALPA}
\texttt{,SEARCH=JPA}
\texttt{,SEARCH=LPA}

Specifies the type of search CSVQUERY is to perform.

\texttt{JPALPA} (the default) causes CSVQUERY to search the caller's job pack area. If the search fails, CSVQUERY searches the link pack area.

\texttt{JPA} causes CSVQUERY to search only the caller's job pack area.

\texttt{LPA} causes CSVQUERY to search only the link pack area.

\texttt{,SEARCHMINOR=NO}
\texttt{,SEARCHMINOR=YES}

Specifies whether to search for minor entry points. \texttt{SEARCHMINOR} is an optional parameter.

\texttt{SEARCHMINOR=NO} specifies that CSVQUERY is not to search for minor entry points. \texttt{NO} is the default.

\texttt{SEARCHMINOR=YES} specifies that CSVQUERY is to search for minor entry points. CSVQUERY locates the minor entry point closest to the address specified on the \texttt{INADDR} parameter. Because the search is for the closest, CSVQUERY must check all entries.

\texttt{,OUTLENGTH=length}

Specifies an optional fullword variable where CSVQUERY is to return the length of the module that it has located. The length returned is the number of bytes used to contain the module. This size may be different depending on whether the module was loaded from a PDS or a PDSE. If there is more than one extent, the length is the sum of all the extents.
If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the length of the module that contains the major entry point associated with the minor entry point.

If the module is a program object bound with the FETCHOPT=NOPACK option, the length value returned was rounded to the fullpage-multiple area that is obtained with GETMAIN to hold the program object. If the program object is bound with the FETCHOPT=PACK option, the length value returned is the size indicated in the directory entry. For more information, see z/OS MVS Program Management: User's Guide and Reference and z/OS MVS Program Management: Advanced Facilities.

,OUTLENGTH64=length64
   Specifies an optional doubleword variable where CSVQUERY is to return the length of the module that it has located. The length returned is the number of bytes used to contain the module. This size may be different depending on whether the module was loaded from a PDS or a PDSE. If there is more than one extent, the length is the sum of all the extents.

If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the length of the module that contains the major entry point associated with the minor entry point.

,OUTEPNM=entryname
   Specifies an optional eight-character variable where CSVQUERY is to return the name of the entry point of the module. When you specify OUTEPNM with INADDR, CSVQUERY returns the module's major entry point name in entryname.

,OUTEPTKN=outeptkn
   Specifies an optional 8-character variable where CSVQUERY returns the output entry point token. Use this token as the input entry point token (INEPTKN) on subsequent invocations of CSVQUERY for the same module.

,OUTEPA=entry addr
   Specifies an optional fullword variable where CSVQUERY is to return the address of the entry point of the module. When you specify OUTEPA with INADDR, CSVQUERY returns the module's major entry point address in entry addr.

,OUTEPA64=entry addr64
   Specifies an optional doubleword variable where CSVQUERY is to return the address of the entry point of the module. When you specify OUTEPA with INADDR, CSVQUERY returns the module's major entry point address in entry addr.

,OUTMJNM=major name
   Specifies an optional eight-character variable where CSVQUERY returns the major name (which is not an alias name) of the module.

,OUTLOADPT=outloadpt
   Specifies an optional fullword variable where CSVQUERY is to return the module's load address.

   If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the load address of the module that contains the major entry point associated with the minor entry point.

,OUTLOADPT64=outloadpt64
   Specifies an optional doubleword variable where CSVQUERY is to return the module's load address.
## CSVQUERY macro

If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the load address of the module that contains the major entry point associated with the minor entry point.

### OUTSP=\texttt{subpool}

Specifies an optional one-byte variable where CSVQUERY returns the subpool number of the module.

If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the subpool number of the module that contains the major entry point associated with the minor entry point.

### OUTATTR1=\texttt{attr1}

Specifies an optional one-byte variable where CSVQUERY returns the attributes of the module.

The bit settings have the following meanings:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning When Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>End-of-memory deletion</td>
</tr>
<tr>
<td>1</td>
<td>Loaded-to-global</td>
</tr>
<tr>
<td>2</td>
<td>Reentrant</td>
</tr>
<tr>
<td>3</td>
<td>Serially reusable</td>
</tr>
<tr>
<td>4</td>
<td>Not loadable only</td>
</tr>
<tr>
<td>5</td>
<td>Overlay format</td>
</tr>
<tr>
<td>6</td>
<td>Alias</td>
</tr>
<tr>
<td>7</td>
<td>Not part of the programming interface</td>
</tr>
</tbody>
</table>

### OUTATTR2=\texttt{attr2}

Specifies an optional one-byte variable where CSVQUERY returns the attributes of the module.

The bit settings have the following meanings:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning When Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Authorized library</td>
</tr>
<tr>
<td>1</td>
<td>Authorized program</td>
</tr>
<tr>
<td>2</td>
<td>AMODE ANY</td>
</tr>
<tr>
<td>3</td>
<td>AMODE 31</td>
</tr>
<tr>
<td>5</td>
<td>Dynamic LPA module</td>
</tr>
<tr>
<td>6</td>
<td>Page protected (only valid for dynamic LPA modules)</td>
</tr>
<tr>
<td>7</td>
<td>AMODE 64</td>
</tr>
</tbody>
</table>

### OUTATTR3=\texttt{attr3}

Specifies an optional one-byte variable where CSVQUERY returns the attributes of the module.

The bit settings have the following meanings:

<table>
<thead>
<tr>
<th>Bit</th>
<th>Meaning When Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Resident above 16 megabytes</td>
</tr>
<tr>
<td>1</td>
<td>Job pack area resident</td>
</tr>
<tr>
<td>2</td>
<td>PLPA resident</td>
</tr>
<tr>
<td>3</td>
<td>MLPA resident</td>
</tr>
<tr>
<td>4</td>
<td>FLPA resident</td>
</tr>
<tr>
<td>5</td>
<td>CSA resident</td>
</tr>
<tr>
<td>6-7</td>
<td>Not part of the programming interface</td>
</tr>
</tbody>
</table>
### OUTXATTR1=xattr

Specifies an optional eight-byte variable where CSVQUERY returns extended attributes of the module.

The bit settings have the following meanings:

<table>
<thead>
<tr>
<th>BYTE</th>
<th>BIT</th>
<th>Meaning When Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>Not part of the programming interface</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>Not part of the programming interface</td>
</tr>
<tr>
<td>2</td>
<td>1..</td>
<td>A RACF basic program</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>A RACF main program</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Not part of the programming interface</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>Not part of the programming interface</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Not part of the programming interface</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>Not part of the programming interface</td>
</tr>
</tbody>
</table>

### OUTVALID=valid

Specifies an optional fullword variable that indicates whether the returned output fields contain valid data. If the bit is set to 1, the corresponding field is valid. Otherwise, the bit is 0. If the return code of the CSVQUERY macro is 0, the validity bits for all requested output are on.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Valid Field When Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OUTLENGTH</td>
</tr>
<tr>
<td>1</td>
<td>OUTEPA</td>
</tr>
<tr>
<td>2</td>
<td>OUTEPNM</td>
</tr>
<tr>
<td>3</td>
<td>OUTMJNM</td>
</tr>
<tr>
<td>4</td>
<td>OUTSP</td>
</tr>
<tr>
<td>5</td>
<td>OUTATTR1</td>
</tr>
<tr>
<td>6</td>
<td>OUTATTR2</td>
</tr>
<tr>
<td>7</td>
<td>OUTATTR3</td>
</tr>
<tr>
<td>8</td>
<td>OUTLOADPT</td>
</tr>
<tr>
<td>9</td>
<td>OUTPDATTA3</td>
</tr>
<tr>
<td>10</td>
<td>OUTPDATTA4</td>
</tr>
<tr>
<td>11</td>
<td>OUTPDATTA5</td>
</tr>
<tr>
<td>12</td>
<td>OUTPDATTA6</td>
</tr>
<tr>
<td>13</td>
<td>OUTPDATTA7</td>
</tr>
<tr>
<td>14</td>
<td>OUTPDATTA8</td>
</tr>
<tr>
<td>15</td>
<td>OUTPDATTA9</td>
</tr>
<tr>
<td>16</td>
<td>OUTPDATTA10</td>
</tr>
<tr>
<td>17</td>
<td>OUTPDATTA11</td>
</tr>
<tr>
<td>18</td>
<td>OUTPDATTA12</td>
</tr>
<tr>
<td>19</td>
<td>OUTPDATTA13</td>
</tr>
<tr>
<td>20</td>
<td>OUTPDATTA14</td>
</tr>
<tr>
<td>21</td>
<td>OUTPDATTA15</td>
</tr>
<tr>
<td>22</td>
<td>OUTPDATTA16</td>
</tr>
<tr>
<td>23</td>
<td>OUTPDATTA17</td>
</tr>
<tr>
<td>24</td>
<td>OUTPDATTA18</td>
</tr>
<tr>
<td>25</td>
<td>OUTPDATTA19</td>
</tr>
<tr>
<td>26</td>
<td>OUTPDATTA20</td>
</tr>
<tr>
<td>27</td>
<td>OUTPDATTA21</td>
</tr>
<tr>
<td>28</td>
<td>OUTPDATTA22</td>
</tr>
<tr>
<td>29</td>
<td>OUTPDATTA23</td>
</tr>
<tr>
<td>30</td>
<td>OUTPDATTA24</td>
</tr>
<tr>
<td>31</td>
<td>OUTPDATTA25</td>
</tr>
</tbody>
</table>

### OUTPDATA=outpdata

Specifies the name, (RS-type), or address in register (2)-(12), of an optional 16 character output variable containing the provider data.

### OUTPID=outpid

Specifies an optional char(4) variable where CSVQUERY returns a string representing the loading service (provider) that loaded the module. The values mean the following:
**CSVQUERY macro**

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>'UNK '</td>
<td>Unknown provider</td>
</tr>
<tr>
<td>'LPA'</td>
<td>LPA</td>
</tr>
<tr>
<td>'PGMF'</td>
<td>Program fetch</td>
</tr>
<tr>
<td>'LLAF'</td>
<td>LLA</td>
</tr>
<tr>
<td>'AOSL'</td>
<td>AOS loader</td>
</tr>
<tr>
<td>'JPA'</td>
<td>JPA</td>
</tr>
</tbody>
</table>

`OUTDIAG=outdiag`  
Specifies the name, (RS-type), or address in register (2)-(12), of an optional 4 character output variable containing the diagnostic data.

`OUTRTID=outrtid`  
Specifies the name, (RS-type), or address in register (2)-(12), of an optional 2 character output variable that, as of z/OS version 1 release 12, does not contain valid information.

`OUTXTLST=xtlst`  
Specifies an optional 136-byte area where CSVQUERY is to return the length and load point information for each segment of the module that it has located. The first four bytes in the area should be initialized to 16 which is the total number of entries which can be returned.

On output, the second four bytes of the area contains the number of 8-byte entries that follow. Each 8-byte entry which follows consists of:
- the load point for that extent as a 4-byte address and
- the length, expressed in bytes, of that extent as 4-byte length.

If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the length(s) of the module that contains the major entry point associated with the minor entry point.

`OUTXTLST64=xtlst64`  
Specifies an optional 264-byte area where CSVQUERY is to return the length and load point information for each segment of the module that it has located. The first four bytes in the area should be initialized to 16 which is the total number of entries which can be returned.

On output, the second four bytes of the area contains the number of 16-byte entries that follow. Each 16-byte entry which follows consists of:
- the load point for that extent as an 8-byte address and
- the length, expressed in bytes, of that extent as 8-byte length.

If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the length(s) of the module that contains the major entry point associated with the minor entry point.

`OUTPATHNAME=opathname`  
Specifies the name, (RS-type), or address in register (2)-(12), of an optional 1026 character output area that is to contain the path name associated with the CDE located by CSVQUERY. Note that this output area might not contain the full path name. The file name returned represents the name that was passed to the file system. Frequently this name is appended to the current home directory, but the home directory will not be returned by CSVQUERY. If the returned name starts with a "/", then it is the full pathname. The first 2–bytes of the area contain the path name length, followed by a path name of up to 1024 characters. A path name length of 0 in the first 2 bytes indicates that there is no path name associated with this CDE.
OUTDSKEY=outdskey
Specifies an optional 8-character output area to contain the key of the data set, which is associated with the CDE that is located by CSVQUERY. Note that the format of this key is not part of the programming interface. A value of zero indicates that the data set key is not available. The validity bit is on whenever the parameter is successfully processed.

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

IMPLIED_VERSION
It is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default. Code or use the default IMPLIED_VERSION with caution in the list form of the MACRO. It could result in storage overlays if parameters are coded on the execute form of the macro which requires a longer parameter list.

MAX
If you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.

The CSVQUERY macro supports multiple versions. The following macro key list contains the version level in which the key was first introduced. When specifying PLISTVER, be sure that it is at least as high as the highest version number of all the keys being used. Explicitly coding a specific plistver value on the list form of the macro which generates a shorter parameter list than that required by the parameters coded on the execute form can result in overlays of storage.

0 Version 0 introduces the following parameters:
  INADDR  OUTTEPA  OUTVALID
  INEPNAME  OUTEPNM  SEARCH
  OUTATTR1  OUTLENGTH  SEARCHMINOR
  OUTATTR2  OUTMJNM
  OUTATTR3  OUTSP

1 Version 1 introduces the following parameter:
  • OUTLOADPT

2 Version 2 introduces the following parameters:
  • INEPTKN
  • OUTEPTKN
  • OUTPDATA
  • OUTPID
The CSVQUERY macro introduces the following parameters:

Version 3 introduces the following parameters:
- OUTDIAG
- OUTRTID
- OUTXTLST

Version 5 introduces the following parameters:
- INADDR64
- OUTLENGTH64
- OUTEPA64
- OUTLOADPT64
- OUTXATTR1
- OUTXTLST64

Version 6 introduces the following parameter:
- OUTPATHNAME

Version 7 introduces the following parameter:
- OUTDSKEY

To code: Specify in this input parameter one of the following values:
- IMPLIED_VERSION
- MAX
- A decimal value in the range of 0 - 6

RETCODE=retcode
Specifies the location where the system is to store the return code. The return code is also in GPR 15. If you specify a storage location, it must be on a fullword boundary.

MF=S
Specifies the standard form of CSVQUERY. The standard form places the parameters into an inline parameter list.

Return and reason codes
When control returns from CSVQUERY, GPR 15 (and retcode, if you coded RETCODE) contains one of the following decimal return codes:

<table>
<thead>
<tr>
<th>Decimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>CSVQUERY retrieves all the requested information.</td>
</tr>
<tr>
<td>04</td>
<td>CSVQUERY locates the specified module, but at least one requested output field is not valid.</td>
</tr>
<tr>
<td>08</td>
<td>CSVQUERY cannot locate the specified module.</td>
</tr>
<tr>
<td>12</td>
<td>CSVQUERY cannot obtain the lock(s) needed to process the request.</td>
</tr>
<tr>
<td>16</td>
<td>CSVQUERY encounters an unexpected error.</td>
</tr>
<tr>
<td>20</td>
<td>The requested function is not available on the system on which CSVQUERY is issued.</td>
</tr>
</tbody>
</table>

CSVQUERY—List form
Use the list form of the CSVQUERY macro to construct a nonexecutable parameter list.
CSVQUERY macro

Syntax

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

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

One or more blanks must follow CSVQUERY.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=listver

Default: IMPLIED_VERSION

plistver: 0 - 6

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 macro with the following exceptions:

```
MF=(L,list_addr)
MF=(L,list_addr,attr)
```

Specifies the list form of the CSVQUERY macro. list_addr defines the area that the system is to use for the parameter list.

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

CSVQUERY—Execute form

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

Syntax

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

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

One or more blanks must precede CSVQUERY.
CSVQUERY macro

CSVQUERY

b

One or more blanks must follow CSVQUERY.

Valid parameters

INEPNAME=entryname
INEPTKN=ineptkn
INADDR=ptr name
INADDR64=ptr name64

,SEARCH=JPALPA
,SEARCH=JPA
,SEARCH=LPA

,SEARCHMINOR=NO
,SEARCHMINOR=YES

,OUTLENGTH=length
,OUTLENGTH64=length64
,OUTEPNM=entryname
,OUTEPTKN=outeptkn
,OUTEPA=entry addr
,OUTEPA64=entry addr64
,OUTMJNM=major name
,OUTLOADPT=outloadpt
,OUTLOADPT64=outloadpt64
,OUTSP=subpool
,OUTATTR1=attr1
,OUTATTR2=attr2
,OUTATTR3=attr3
,OUTDIAG=outdiag
,OUTRTID=outrtid
,OUTXATTR=xattr
,OUTVALID=valid
,OUTPDATA=outpdata

Default: JPALPA
Default: NO

length: RS-type address or register (2) - (12).
length64: RS-type address or register (2) - (12).
entry name: RS-type address or register (2) - (12).
ptr name: RS-type address or register (2) - (12).
ptr name64: RS-type address or register (2) - (12).
ptr addr: RS-type address or register (2) - (12).
ptr addr64: RS-type address or register (2) - (12).
major name: RS-type address or register (2) - (12).
outloadpt: RS-type address or register (2) - (12).
outloadpt64: RS-type address or register (2) - (12).
subpool: RS-type address or register (2) - (12).
attr1: RS-type address or register (2) - (12).
attr2: RS-type address or register (2) - (12).
attr3: RS-type address or register (2) - (12).
outdiag: RS-type address or register (2) - (12).
outrtid: RS-type address or register (2) - (12).
xattr: RS-type address or register (2) - (12).
valid: RS-type address or register (2) - (12).
outdata: RS-type address or register (2) - (12).
Parameters

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

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

Specifies the execute form of the CSVQUERY macro. `list addr` defines the area that the system uses for the parameter list.

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

`NOCHECK` specifies that the system is to check only parameters that you specified.

CSVQUERY—Modify form

The modify form of the CSVQUERY macro can refer to and modify the parameter list constructed by the list form of the macro.

Syntax

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

```
name
b
```

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

One or more blanks must precede CSVQUERY.
CSVQUERY macro

Valid parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INEPNAME</td>
<td>entryname: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>INEPKTN</td>
<td>ineptkn: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>INADDR</td>
<td>ptr name: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>INADDR64</td>
<td>ptr name64: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SEARCH</td>
<td>Default: JPALPA</td>
</tr>
<tr>
<td>,REAL</td>
<td></td>
</tr>
<tr>
<td>,REAL64</td>
<td></td>
</tr>
<tr>
<td>,REALX</td>
<td></td>
</tr>
<tr>
<td>,REALXX</td>
<td></td>
</tr>
<tr>
<td>,REALX64</td>
<td></td>
</tr>
<tr>
<td>,REALXX64</td>
<td></td>
</tr>
<tr>
<td>,REALXXX</td>
<td></td>
</tr>
<tr>
<td>,REALXXX64</td>
<td></td>
</tr>
<tr>
<td>,REALXXS</td>
<td></td>
</tr>
<tr>
<td>,REALXXS64</td>
<td></td>
</tr>
<tr>
<td>,REALXXSXX</td>
<td></td>
</tr>
<tr>
<td>,REALXXSXX64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXS</td>
<td></td>
</tr>
<tr>
<td>,REALXXXS64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXX</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXX64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXS</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXS64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSS</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSS64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSSXX</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSSXX64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXS</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXS64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXX</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXX64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXXS</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXXS64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXXSXX</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXXSXX64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXXSXXS</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXXSXXS64</td>
<td></td>
</tr>
<tr>
<td>,REALXXXSXXSXXSXXSXX</td>
<td></td>
</tr>
</tbody>
</table>

Parameters

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

- **MF=(M,list addr)**
  - Specifies the modify form of the CSVQUERY macro. list addr defines the area that the system uses for the parameter list.
  - COMPLETE specifies that the system is to check for required parameters and supply optional parameters that are not specified. COMPLETE is the default.
  - NOCHECK specifies that the system is to check only parameters that you specified.
Chapter 85. DELETE — Relinquish control of a load module

Description

The DELETE macro cancels the effect of a previous LOAD macro. If DELETE cancels the only outstanding LOAD request for the module, and no other requirements exist for the module, the virtual storage occupied by the load module is released and is available for reassignment by the control program.

In the case of nonreentrant and nonreusable modules loaded multiple times, the order of processing occurs in last-loaded first-deleted order. For example, if Program A loads module LOADMODA, then calls Program B, which also loads LOADMODA, then issues a DELETE against LOADMODA, the copy of the load module to be deleted is the one associated with Program B. At this point, a copy of LOADMODA will still exist. The next DELETE request against LOADMODA will delete that copy, regardless of whether Program A or Program B issues the request.

The entry name specified in the DELETE macro must be the same as that specified in the LOAD macro that brought the load module into storage. Also, the DELETE macro must be issued by the same task that issued the LOAD macro.

Any module loaded by a task will not be removed from virtual storage until the DELETE macro is issued or end of task is reached. In addition, any module loaded by a system task will not be removed from virtual storage until a DELETE macro is issued by a system task or end of task is reached.

Environment

The requirements for the caller are:

Minimum authorization: Problem state and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 24- or 31- or 64-bit
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Control parameters must be in the primary address space.

Programming requirements

- The entry name specified in the DELETE macro must be the same as that specified in the LOAD macro that brought the load module into storage.
- The DELETE macro must be issued by the same task that issued the LOAD macro.

Restrictions

None.

Input register information

None.
DELETE macro

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Address of the name or list entry that was supplied through the EP or DE keyword.</td>
</tr>
<tr>
<td>1-14</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>15</td>
<td>Return code.</td>
</tr>
</tbody>
</table>

Syntax

The DELETE macro is written as follows:

```
name name
\ /\ /\ /\ /
   One or more blanks must precede DELETE.
   DELETE
   \ /\ /\ /\ /
      One or more blanks must follow DELETE.

EP=entry name
EPLOC=entry name addr
DE=list entry addr
,RELATED=value

entry name: Symbol.
entry name addr: RX-type address, or register (0) or (2)- (12).
list entry addr: RX-type address, or register (0) or (2)- (12).

value: any valid macro keyword specification.
```

Parameters

The parameters are explained as follows:

```
EP=entry name
EPLOC=entry name addr
DE=list entry addr
,RELATED=value

Specifies the entry name, the address of the entry name, or the address of a
62-byte list entry for the entry name that was constructed using the BLDL
macro. If you code EPLOC, pad the name to eight bytes, if necessary.

,RELATED=value

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

The RELATED parameter is available on macros that provide opposite services
(for example, ATTACH/DETACH, GETMAIN/FREEMAIN, and LOAD/DELETE),
and on macros that relate to previous occurrences of the same macros (for
example, CHAP and ESTAE).
```
ABEND codes
None.

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Successful completion of requested function</td>
</tr>
<tr>
<td>04</td>
<td>Requested module was not in storage, or an attempt was made to delete a system module by a caller not authorized to do so</td>
</tr>
</tbody>
</table>

Example
Remove a module (PGMTOVLY) from virtual storage.
DELETE EP=PGMTOVLY
DELETE macro
Chapter 86. DEQ — Release a serially reusable resource

Description

The DEQ macro releases control of one or more serially reusable resources from the active task. A task ends abnormally if it either requests an unconditional release of a resource it does not control, or issues a request that contains incorrect parameters.

When you use DEQ to release control of a resource obtained through the ENQ macro, certain parameters on DEQ must match the parameters on the ENQ that assigned control to that resource. Similarly, when you use DEQ to release control of a resource obtained through the RESERVE macro, certain parameters on DEQ must match the parameters on the RESERVE that assigned control to that resource. In the cases where the parameters must match, the parameter descriptions note that fact.

An explanation of how to use the DEQ macro to serialize access to resources appears in z/OS MVS Programming: Assembler Services Guide.

Environment

The requirements for callers of DEQ are:

Minimum authorization: Problem state with any PSW key.
Dispatchable unit mode: Task
Cross memory mode:
For LINKAGE=SVC: PASN=HASN=SASN
For LINKAGE=SYSTEM: Any PASN, Any HASN, Any SASN
For LINKAGE=SYSTEM with RMC=STEP: PASN=HASN, Any SASN
AMODE: 24- or 31- or 64-bit
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

The caller cannot have an EUT FRR established.

Input register information

Before issuing the DEQ 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
DEQ macro

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=HAVE, if all return codes for the resources named in the DEQ macro are 0, register 15 contains 0. If any of the return codes are not 0, register 15 contains the address of a storage area containing the return codes.
   • Otherwise: Used as a work register by the system.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 DEQ macro is written as follows:

```
name

b

DEQ

b

( (qname addr

, ,rname addr

, ,rname length

, ,STEP

, ,SYSTEM

, ,SYSTEMS
```

name: symbol. Begin name in column 1.

One or more blanks must precede DEQ.

One or more blanks must follow DEQ.

qname addr: A-type address, or register (2) - (12).

rname addr: A-type address, or register (2) - (12).

rname length: symbol, decimal digit, or register (2) - (12).

Note: rname length must be coded if a register is specified for rname addr.

Default: STEP
DEQ macro

Parameters

The parameters are explained as follows.

(  Specifies the beginning of the resource description.

qname addr
  Specifies the address of an 8-character name. The name can contain any valid hexadecimal characters. The qname must be the same name specified for the resource in an ENQ or RESERVE macro. Authorized programs should use a restricted qname (as described under Minimum authorization in the Environment section of this chapter) to prevent interference from unauthorized programs.

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

, rname addr
  Specifies the address of the name used together with qname and scope to represent the resource acquired by a previous ENQ or RESERVE macro. The name must be from 1 to 255 bytes long, can be qualified, and can contain any valid hexadecimal characters. The name must be the same name specified for the resource in an ENQ or RESERVE macro.

, rname length
  Specifies the length of the name. The length must have the same value as specified in the previous ENQ or RESERVE macro. If you omit this parameter, the system uses the assembled length of the name. You can specify a value between 1 and 255 to override the assembled length, or you may specify a value of 0. If you specify 0, the length of the name must be contained in the first byte at the rname addr.
DEQ macro

,SYSTEMS
   Specifies the scope of the resource. If you used the ENQ macro to obtain
   control of the resource, the scope you specify on DEQ must match the scope
   specified on that ENQ. If you used the RESERVE macro to obtain control of the
   resource, you must specify SYSTEMS as the scope on DEQ.
)
   Specifies the end of the resource description.

Notes on specifying multiple resources on one DEQ request:
- Within a single set of parentheses, you can repeat the qname addr, rname addr,
  type of control, mame length, and the scope until there is a maximum of 255
  characters, including the parentheses.
- The following parameters apply to all the resources you specify on the request:
  RET and RNL.

,RET=None
,RET=HAVE
   HAVE specifies that the request for releasing the resources named in DEQ is to
   be honored only if the active task has been assigned control of the resources. A
   return code is set if the resource is not held. NONE specifies an unconditional
   request to release all the resources. RET=NONE is the default. The active task
   ends abnormally if it has not been assigned control of the resources.

,UCB=ucb addr
   Specifies the address of a fullword that contains the address of a UCB for a
   reserved device that is now being released. This parameter is used to release a
   device reserved with the RESERVE macro and is valid only with a scope of
   SYSTEMS. The UCB parameter is optional.

   Note: The UCB keyword might contain a UCB address for a UCB that resides
   in storage above or below 16 megabytes. If the UCB address might point
   to a UCB above 16 megabytes, you must also specify LOC=ANY.

,LOC=BELOW
,LOC=ANY
   Specifies the location of the input UCB address. ANY specifies that the input
   UCB address is to be treated as a 31-bit address. BELOW specifies that the
   input UCB address is to be treated as a 24-bit address. The default is
   LOC=BELOW.

,RNL=YES
,RNL=NO
   Specifies whether the system is to perform RNL processing, which might
   change the scope value of a resource. You must specify the same RNL option
   as you used in the ENQ macro that requested the resource. The default is
   RNL=YES.

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

,LINKAGE=SVC
,LINKAGE=SYSTEM
   Specifies the type of linkage the caller is using to invoke the DEQ service.
For LINKAGE=SVC, the linkage is through an SVC instruction. This linkage is valid only when the caller is in primary mode and the primary, home, and secondary address spaces are the same.

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

The default is LINKAGE=SVC.

**ABEND codes**

For only unconditional requests, the caller might encounter abend code X’130’ or X’530’. For unconditional and conditional requests, the caller might encounter one of the following abend codes:

- X’230’
- X’330’
- X’430’
- X’730’
- X’830’
- X’930’

See [z/OS MVS System Codes](https://www.ibm.com) for explanations and responses for these codes.

**Return and reason codes**

Return codes are provided by the system only if RET=HAVE is designated. If all of the return codes for the resources named in DEQ are 0, register 15 contains 0. If any of the return codes are not 0, register 15 contains the address of a virtual storage area containing the return codes as shown in Figure 5.
DEQ macro

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

The return codes for the DEQ macro with the RET=HAVE parameter are described in Table 14.

**Table 14. Return Codes for the DEQ Macro with the RET=HAVE Parameter**

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** The system has released the resource.  
**Action:** None. |
| 4                       | **Meaning:** The resource has been requested for the task, but the task has not been assigned control of it. The task continues waiting. (This return code might result if an exit routine, which received control because of an interruption, issued the DEQ macro on behalf of the task.)  
**Action:** None. |
| 8                       | **Meaning:** Control of the resource has not been requested by the active task, or the resource has already been released.  
**Action:** None required. However, you might take some action based on your application. |

**Example 1**

Release control of the resource in Example 1 of ENQ (see Chapter 93, “ENQ — Request control of a serially reusable resource,” on page 539), if it has been assigned to the current task.
Example 2

Unconditionally release control of the resources in Example 2 of ENQ. The length of the name for the first resource is 3 characters and the length of the name for the third resource is 8 characters. Allow the length of the second resource to default to its assembled length.

DEQ (MAJOR4,MINOR4,3,STEP,MAJOR2,MINOR2,,SYSTEM, X
MAJOR3,MINOR3,8,SYSTEMS)

DEQ—List form

Use the list form of the DEQ macro to construct a control program parameter list. The number of qname, rname, and scope combinations in the list form of DEQ must be equal to the maximum number of qname, rname, and scope combinations in any execute form of DEQ that refers to that list form.

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

<table>
<thead>
<tr>
<th>name</th>
<th>name: symbol. Begin name in column 1.</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>One or more blanks must precede DEQ.</td>
</tr>
<tr>
<td>DEQ</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>One or more blanks must follow DEQ.</td>
</tr>
</tbody>
</table>

(  
qname addr  
.name addr  
, rname addr  
, rname length  
,STEP  
,System  
, Systems  
)  

,RET=NONE  
,RET=HAVE  
,UCB=ucb addr  
,LOC=BELOW  

Default: RET=NONE  
Default: LOC=BELOW
DEQ macro

_DEFAULT: RNL=YES

,LOC=ANY

, RNL=YES

, RNL=NO

,RELATED=value

value: any valid macro keyword specification.

, MF=L

Parameters

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

, MF=L

Specifies the list form of the DEQ macro.

DEQ-Execute form

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

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

name

name: symbol. Begin name in column 1.

b

One or more blanks must precede DEQ.

DEQ

One or more blanks must follow DEQ.

( Note: ( and ) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired, then (., ) and all parameters between ( and ) should not be specified. If something in the list is desired, 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).

, mame addr

mame addr: RX-type address, or register (2) - (12).

, mame length

mame length: symbol, decimal digit, or register (2) - (12).

, STEP
DEQ macro

Note: See note opposite ( above.

,RET=NONE
,RET=HAVE

,UCB=ucb addr
ucb addr: RX-type address, or register (2) - (12).
Note: Specify UCB only with SYSTEMS.

,LOC=BELOW
,LOC=ANY

Default: LOC=BELOW

,RNL=YES
,RNL=NO

,RELATED=value
value: any valid macro keyword specification.

,LINKAGE=SVC
,LINKAGE=SYSTEM

DEFAULT: LINKAGE=SVC

,MF=(E,list addr)
list addr: RX-type address, or register (1) - (12).

Parameters

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

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

list addr specifies the area that the system uses to contain the parameters.
DEQ macro
Chapter 87. DETACH — Detach a subtask

Description

The DETACH macro removes from the system a subtask created using the ATTACH (or ATTACHX) macro with the ECB or ETXR parameters. Subtasks created using the ATTACH macro without specifying the ECB or ETXR parameters are automatically removed by the system when they terminate. If a task attaches a subtask with the ECB or ETXR parameters, the originating task must detach the subtask before terminating.

You can issue a DETACH macro only for subtasks created by the active task.

Environment

The requirements for the caller are:

Minimum authorization: Problem state, and any PSW key.
Dispatchable unit mode: Task
Cross memory mode: PASN=HASN=SASN
AMODE: 24- or 31- or 64-bit
ASC mode: Primary or AR
Interrupt status: Enabled for I/O and external interrupts
Locks: The caller may not hold any locks.
Control parameters: Must be in the primary address space.

Programming requirements

If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before you issue DETACH.

Restrictions

- Failure to remove subtasks created using the ATTACH macro with the ECB or ETXR parameters causes the originating task and all of its subtasks to terminate abnormally.
- Detaching a terminated subtask that was created without the ECB or ETXR parameters will cause the originating task and all its subtasks to terminate abnormally.
- Detaching a task that has not yet terminated will cause that task and all its subtasks (but not the originating task) to terminate abnormally.
- The caller cannot have an EUT FRR established.

Input register information

Before issuing the DETACH 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
</tbody>
</table>
DETACH macro

15    Return code

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 DETACH macro is written as follows:

```
name name

name: Symbol. Begin name in column 1.

b
b

One or more blanks must precede DETACH.

DETACH

One or more blanks must follow DETACH.

```

```
tcb addr
tcb addr: Symbol, RX-type address, or register (1) or (2) - (12).

,STAE=NO
,STAE=YES

Default: STAE=NO

,RELATED=value

```

Parameters

The parameters are explained as follows:

```
tcb addr

Specifies the address of a fullword on a fullword boundary containing the address of the task control block for the subtask to be removed from the system.

,STAE=NO
,STAE=YES

Specifies whether the ESTAE-type routine (STAI, ESTAI, STAE, ESTAE)
established by the subtask is to receive control or whether previously established ESTAE-type routines existing for the subtasks are to receive control.

If you specify STAE=YES, any ESTAE-type routines associated with the detached task will receive control if the task is detached while active.

If you specify STAE=NO, only the ESTAE-type routines that were established through the ATTACH, ESTAE, or ESTAEX macros, with TERM=YES, will receive control in this event.

When an ESTAE-type routine gains control as a result of a DETACH, no retry is allowed even if one is requested in the routine. For more information about recovery processing, refer to "z/OS MVS Programming: Assembler Services Guide".

\texttt{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.

\textbf{ABEND codes}

The caller of DETACH might receive one of the following ABEND codes:

<table>
<thead>
<tr>
<th>ABEND Code</th>
<th>Associated Reason Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>X'13E'</td>
<td>None</td>
</tr>
<tr>
<td>X'23E'</td>
<td>0, 4, 8</td>
</tr>
<tr>
<td>X'33E'</td>
<td>None</td>
</tr>
<tr>
<td>X'43E'</td>
<td>None</td>
</tr>
<tr>
<td>X'53E'</td>
<td>None</td>
</tr>
</tbody>
</table>

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

\textbf{Return and reason codes}

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                      | \textbf{Meaning:} Successful completion.  
\textbf{Action:} None. |
| 04                      | \textbf{Meaning:} Environmental error. An incomplete subtask was detached with STAE=YES specified; DETACH processing successfully completed.  
\textbf{Action:} None required. However, you might take some action based upon your application. |

\textbf{Example 1}

Remove the subtask from the address space. The address of the TCB is in the fullword labeled SAVEWORD.

\texttt{DETACH SAVEWORD}
DETACH macro

Example 2

In addition to removing the subtask from the address space, give control to the most recent STAE exit established by the subtask if the subtask has not yet been terminated.

DETACH (1),STAE=YES
Chapter 88. DIV — Data-in-virtual

Description

The DIV macro establishes a window in an address space, data space, or hiperspace to reference and update data from a data-in-virtual object without actually issuing I/O instructions. The data-in-virtual object can be a VSAM linear data set or a nonshared standard hiperspace.

The DIV macro accesses a data object on permanent storage through paging I/O. Data-in-virtual maps the object onto a single virtual address range so your program can view it as beginning at a virtual location and occupying a consecutive virtual address range.

If the window is in an address space or a data space, use assembler instructions to access data. If the window is in a hiperspace, use the HSPSERV macro to access data in 4K-byte blocks.

The DIV macro performs the following services:

<table>
<thead>
<tr>
<th>Service</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDENTIFY</td>
<td>Identifies you as a user of a data-in-virtual object.</td>
</tr>
<tr>
<td>ACCESS</td>
<td>Provides access to the data-in-virtual object.</td>
</tr>
<tr>
<td>MAP</td>
<td>Makes the data-in-virtual object addressable through your virtual window.</td>
</tr>
<tr>
<td>RESET</td>
<td>Releases changes made in your window since the last SAVE operation.</td>
</tr>
<tr>
<td>SAVE</td>
<td>Saves changed data that is in your window.</td>
</tr>
<tr>
<td>SAVELIST</td>
<td>Returns the addresses of the first and last changed pages in each range of changed pages within the window.</td>
</tr>
<tr>
<td>UNMAP</td>
<td>Eliminates the correspondence between the data-in-virtual object and your virtual window.</td>
</tr>
<tr>
<td>UNACCESS</td>
<td>Eliminates your access to the data-in-virtual object.</td>
</tr>
<tr>
<td>UNIDENTIFY</td>
<td>Ends your use of the data-in-virtual object.</td>
</tr>
</tbody>
</table>

The services of data-in-virtual execute synchronously, that is, control does not return from the DIV macro until the service is completed. Thus, before you can successfully invoke a service, the previous service must be complete.

For guidance information on the use of data-in-virtual, see z/OS MVS Programming Assembler Services Guide.

Environment

The requirements for the caller are:

- **Minimum authorization:** Problem state and any PSW key.
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** 31-bit
- **ASC mode:** Primary or access register (AR)
**DIV macro**

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

**Programming requirements**

Before using the DIV macro, the caller must first create either a linear data set object or a hiperspace object. The user must also supply a standard 72-byte save area.

**Restrictions**

- When you attach a new task, you cannot pass ownership of a mapped virtual storage window to the new task. That is, you cannot use the ATTACH or ATTACHX keywords GSPV and GSPL to pass the mapped virtual storage.
- While your program is in cross-memory mode, your program cannot invoke data-in-virtual services; however, your program can reference and update data in a mapped virtual storage window.
- The task that obtains the ID (through DIV IDENTIFY) is the only one that can issue other DIV services for that ID.
- When you identify a data-in-virtual object using the IDENTIFY service, you cannot request a checkpoint until you invoke the corresponding UNIDENTIFY service.
- When you use DIV with the IARVSE 3R macro to share data in virtual storage, you must follow several requirements; see the chapter about sharing data through IARVSE 3R in [z/OS MVS Programming: Assembler Services Guide](https://www.ibm.com). The DIV macro does not support VSAM extended format linear data sets for use as a DIV object for which the size is greater than 4GB.

**Input register information**

The DIV macro is sensitive to the SYSSTATE macro with the OSREL parameter

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>The address of an 18-word save area</td>
</tr>
</tbody>
</table>

**Output register information**

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR 15 contains a nonzero return code; otherwise, used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
</table>
DIV macro

0 and 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

- By using the DIV macro, you might reduce the amount of I/O. The SAVELIST service additionally improves performance of the application when it is necessary to inspect and verify data only in pages that have changed.
- Using LOCVIEW=MAP on a DIV ACCESS request degrades performance. Use LOCVIEW=None request whenever possible. You can use LOCVIEW=MAP request for small data objects without significant performance loss.
- Using RETAIN=YES on a DIV UNMAP request can degrade performance. Using RETAIN=YES causes the system to read more pages from the object.

Syntax

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

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

b
DIV
b
```

Valid parameters:

(Underlined parameters are those that you must specify.)

- **IDENTIFY ID, TYPE, DDNAME or STOKEN**
- **ACCESS ID, MODE, SIZE, LOCVIEW**
- **MAP ID, AREA, OFFSET, SPAN, STOKEN, RETAIN, PFCOUNT**
- **RESET ID, OFFSET, SPAN, RELEASE**
- **SAVE ID, OFFSET, SPAN, SIZE, STOKEN, LISTADDR, LISTSIZE, MF**
- **SAVELIST ID, LISTADDR, LISTSIZE, MF**
- **UNMAP ID, AREA, RETAIN, STOKEN**
- **UNACCESS ID**
- **UNIDENTIFY ID**

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

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

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

- **,LISTADDR=listaddr**
  
  `addr`: RX-type address, or register (2) - (12).
DIV macro

,LISTSIZE=listsize
    addr: RX-type address, or register (2) - (12).

,LOCVIEW=MAP
    Default: LOCVIEW=NONE

,LOCVIEW=NONE

,MODE=READ
    Default: None

,MODE=UPDATE

,OFFSET=addr
    addr: RX-type address, or register (2) - (12).

,OFFSET=* 
    Default: OFFSET=0

,RETAIN=YES
    Default: RETAIN=NO

,RETAIN=NO

,SIZE=addr
    addr: RX-type address, or register (2) - (12).

,SIZE=* 
    Default: OFFSET=0

,SPAN=addr 
    addr: RX-type address, or register (2) - (12).

,SPAN=* 

,STOKEN=addr
    addr: RX-type address.

,TYPE=DA
    Default: None

,TYPE=HS

,PFCOUNT=nnn
    Default: 0

,RELEASE=YES
    Default: RELEASE=NO

,RELEASE=NO

Parameters

The IDENTIFY, ACCESS, MAP, SAVE, SAVELIST, RESET, UNMAP, UNACCESS, and UNIDENTIFY parameters, which designate the services of the DIV macro, are mutually exclusive. You can select only one. The parameters are explained as follows:

IDENTIFY

Selects the data-in-virtual object (linear data set or hiperspace) that you want to process. When you specify IDENTIFY, you must also specify ID and TYPE. ID specifies the address of an eight-byte field into which the IDENTIFY service returns a unique eight-byte name. When you invoke other data-in-virtual services, you use this identifier, or token, as input. The use of the ID is associated only with your task; that is, all services for this ID must be requested by the same task that obtained the ID.

When the object is a data set, you must also specify TYPE=DA and DDNAME. When the object is a nonshared standard hiperspace, you must specify TYPE=HS and STOKEN.

ACCESS

Requests permission to access a data-in-virtual object. When you specify ACCESS, you must also specify ID and MODE, and you may optionally specify SIZE or LOCVIEW. ID specifies the token which identifies the object you want
DIV macro

to access. If your object is a hiperspace, ACCESS allows either multiple readers or one updater. Therefore, the system does not accept a read request if there is already an updater, and it does not accept an update request if there is any other user currently accessing the same object. You cannot access a hiperspace as a data object if it is, or has been on an access list.

MAP
Establishes addressability to the object in a specified range of virtual storage, called the virtual window. When you specify MAP, you must also specify ID and AREA, and you may optionally specify OFFSET, SPAN, STOKEN, RETAIN, and PFCOUNT. Specify STOKEN when your window is in a data space or a standard hiperspace. If your window is in an address space, your object can be either a linear data set or a nonshared standard hiperspace. If your window is in a data space or a hiperspace, your object can be only a linear data set.

If you specified TYPE=DA, you can issue more than one MAP with different STOKENs. You cannot mix data space and hiperspace maps with address space maps under the same ID at any one time.

RESET
Releases changes made in the window since the last SAVE operation. When you specify RESET, you must also specify ID, and you may optionally specify OFFSET, SPAN, and RELEASE. If the window corresponds to blocks on the object, the current contents of the object will replace the data that has changed in the window when the program next references the window. RESET does not change the object.

Do not specify RESET for a storage range that contains DREF storage.

SAVE
Writes changed pages from the window to the corresponding blocks in the object. When you specify SAVE, you must also specify ID, and you may optionally specify OFFSET, SPAN, SIZE, and STOKEN. The system writes changed pages from the window into the blocks specified by OFFSET and SPAN. SAVE cannot change the size of a hiperspace object.

Do not specify SAVE for a storage range that contains DREF storage.

Optionally, SAVE accepts a user list that the application specifies through the LISTADDR and LISTSIZE parameters. The user list contains information returned by the SAVELIST service. If you specify a user list as input for SAVE, you cannot specify OFFSET and SPAN, and the system saves only those pages specified in the user list.

SAVELIST
Returns the addresses of the first and last changed pages in each range of changed pages within the window. The mapped ranges may be either address spaces, data spaces, or hiperspaces. If more than one data space or hiperspace is mapped onto a DIV object, the selected range must be contained within a single data space or hiperspace.

UNMAP
Terminates a virtual window by removing the correspondence between virtual pages in the window and blocks in the object. After the UNMAP is complete, the contents of the pages depend on the value you specify for RETAIN; the virtual pages in the former window either retain the current view of the object or appear as if they had just been obtained.

When you specify UNMAP, you must also specify ID and AREA, and you may specify RETAIN and STOKEN if the object is a data set and the window is in a data space or a hiperspace. UNMAP has no effect on the object itself and does
not save data from the virtual window. If you want to save the data in the window, invoke SAVE before you invoke UNMAP.

If you issued multiple MAPs with different STOKENs, use STOKEN on UNMAP. If you do not specify STOKEN, the system scans the mapped ranges and unmaps the first range that matches the virtual area regardless of the data space it is in. Issuing UNACCESS or UNIDENTIFY automatically unmaps all mapped ranges.

**UNACCESS**
Relinquishes your permission to read from or write to a data-in-virtual object. When you specify UNACCESS, you must also specify ID, which provides the address of the unique name that was returned by the IDENTIFY service. When you invoke UNACCESS, any outstanding windows for the specified ID are automatically unmapped with an implied RETAIN=NO.

**UNIDENTIFY**
Ends the use of a data-in-virtual object under a previously assigned ID. When you specify UNIDENTIFY, you must also specify ID, which provides the address of the unique name that was returned by the IDENTIFY service. If the object is still accessed or mapped under the specified ID, the system will automatically unaccess and unmap it with an implied RETAIN=NO.

**ID=** addr
Specifies the address of a field in storage where the IDENTIFY service stores a unique eight-byte name that it associates with the object. This name acts as a token and is the output value from the IDENTIFY service. It is a required input value for all the other services.

**AREA=** addr
Specifies the address of a four-byte field in storage containing a pointer to the start of the virtual window. You must specify the AREA parameter when you invoke the MAP and the UNMAP services. The starting address for an UNMAP request must be identical to the starting address of its corresponding MAP request. Address space virtual storage that is occupied by a window must meet the following requirements:
- The window must begin on a 4096-byte (page) boundary and must be a multiple of 4096 bytes long.
- Virtual storage within the window must have been obtained from a single, pageable, private area subpool owned by the task that issued the IDENTIFY.
- The window cannot contain VIO storage.
- Pages within the window cannot be page fixed.

Data space and hiperspace virtual storage that is occupied by a window must meet the following requirements.
- The window must be on a 4096-byte boundary and must be a multiple of 4096 bytes long.
- The data space or hiperspace must be owned or created by the task specifying the MAP service.
- The data space or hiperspace must exist until you specify the UNMAP service for all mapped ranges.
- The specified mapped range must lie within the current bounds of the data space or hiperspace.

**DDNAME=** addr
Specifies the address of a field containing the ddname for the data set object when you specify TYPE=DA on IDENTIFY. The first byte of the field must be
DIV macro

the number of characters in the ddname. The bytes following the first byte must
contain the EBCDIC characters of the ddname itself. The ddname must conform
to the standard syntax for ddnames (one through eight alphameric or national
characters). DDNAME is required when you invoke IDENTIFY with TYPE=DA
for a data set object but is not allowed when you specify TYPE=HS for a
hiperspace object. Do not specify a DDNAME that corresponds to a VSAM
extended format linear data set for which the size is greater than 4GB, because
the DIV macro does not support them for use as a DIV object.

,,LISTADDR=listaddr
    Specifies the address of a 4-byte field that contains a pointer to the user list
    that the caller provides for the SAVELIST service.

,,LISTSIZE=listsize
    Specifies the address of a 4-byte field that contains the number of entries in the
    user list for the SAVELIST service. The size of the list must be a minimum of
    three entries and a maximum of 255 entries, where each entry contains two
    words.

,,LOCVIEW=MAP
,,LOCVIEW=NONE
    Specifies whether the system is to create a local copy of the data-in-virtual
    object. For hiperspace objects, you must specify LOCVIEW=NONE or use the
    default.
    LOCVIEW=MAP specifies that the system is to establish a local copy of the
data set object for the specified range. Use MAP to maintain a consistent view
in the virtual storage window of data on permanent storage in environments
where there are multiple writers or at least one reader and writer at the same
time to the object.
    LOCVIEW=NONE specifies that the system is not to create a local copy of the
object. NONE is the default. Use NONE in environments where there is either a
single writer, OR one or more readers, but not both at the same time.

,,MODE=READ
,,MODE=UPDATE
    Specifies whether the object is being accessed for the purpose of reading or
    updating. If you are using the SAVE service to update an object, specify
    MODE=UPDATE. Otherwise, specify MODE=READ to signify read-only access
to the object. You must specify MODE whenever you specify ACCESS.

,,OFFSET=addr
,,OFFSET=*
    Specifies the beginning of a continuous range of blocks in a data-in-virtual
    object. OFFSET is used with SPAN to define a continuous range of blocks in an
    object. OFFSET designates the location of the first block in the range, and
    SPAN designates how many blocks are in the range. An OFFSET value of zero
designates the first block (the beginning) of an object. The system permits an
OFFSET beyond the current end of the object as long as it remains within the
maximum number of blocks allowed for the object and also within the absolute
limit of (2**20)-1 blocks. If you omit OFFSET or specify OFFSET=*, the system
uses a default OFFSET of zero. You can specify the OFFSET parameter with
MAP, RESET, and SAVE.

,,RETAIN=YES
,,RETAIN=NO
    Determines what data appears in the window when you invoke the MAP
    service, and what data is left in virtual storage when you invoke UNMAP.
DIV macro

When you specify RETAIN=YES with MAP, the data in the virtual range stays the same. The system considers all pages in the range changed. When you specify RETAIN=NO (or use the default) with MAP, data in the object replaces the data in virtual range.

When you specify RETAIN=NO with UNMAP, the data in the virtual range becomes freshly obtained. When you specify RETAIN=YES with UNMAP, the virtual range retains its current view.

\$SIZE=addr
\$SIZE=*

Specifies the address of a field where the system stores the size of the object. The system returns the size in this field whenever you specify SAVE or ACCESS and also specify SIZE. When the system returns control after executing a SAVE, the value that it returns is the minimum number of blocks that must be mapped to ensure that the entire object is mapped. If you omit SIZE or specify SIZE=*, the system does not return the size.

If you specified TYPE=DA for a linear data set object, and you specify SIZE, the macro returns the current size of the object in the four-byte location that SIZE designates.

If you specified TYPE=HS for a hiperspace object, and you specify SIZE, ACCESS returns two sizes in the eight-byte location. The first is the current size of the hiperspace (in 4K byte units), and the second is the maximum size of the hiperspace (also in 4K byte units).

Specify SIZE only when you specify ACCESS or SAVE.

\$SPAN=addr
\$SPAN=*

Specifies the address of a four-byte field containing the number of blocks that are to be processed. Use SPAN only with the MAP, RESET, or SAVE services, which operate only on a range of contiguous blocks. SPAN indicates how many blocks are in the range. It is used with OFFSET, which indicates the first block of the range.

For the RESET and SAVE services, the block range can include noncontiguous mappings of an object. This lets you reset or save several maps in a single DIV macro invocation.

For the MAP service, the block range can extend beyond the end of the object, but it cannot extend beyond the maximum size allowed for the object. You can create a window that exceeds the size of the object. The maximum span allowed is \((2^{20})-1\) blocks.

If you omit SPAN or specify SPAN=*, or if the four-byte field contains zero, the system uses the SPAN default value. For the SAVE and RESET services, the default value is the number of blocks in the object from the specified or defaulted block to the end of the last mapped range. For the MAP service, the default is the current size of the object in blocks, minus the value specified by OFFSET. If the offset value is beyond the end of the object, the span defaults to one when you omit SPAN.

\$STOKEN=addr

Specifies the address of an eight-byte field that identifies a hiperspace or data space. STOKEN is valid only with the IDENTIFY, MAP, SAVE, and UNMAP parameters. Specify STOKEN with MAP to map a linear data set object onto data space or hiperspace virtual storage, or to unmap data space or hiperspace storage.
DIV macro

With MAP, the system maps the permanent object into the data space or
hiperspace that the STOKEN represents. If you do not specify STOKEN, the
mapping applies to the primary address space. With UNMAP, STOKEN
identifies which data space or hiperspace contains the window to be unmapped.

If you specified TYPE=HS for a hiperspace object, specify STOKEN with
IDENTIFY. The system does not verify the STOKEN until you use the
associated ID with ACCESS.

,TYPE=DA
,TYPE=HS

TYPE=DA specifies that your program is using a data definition statement to
identify a VSAM linear data set as the data object. The DIV macro does not
support VSAM extended format linear data sets for use as a DIV object for
which the size is greater than 4GB. TYPE=HS specifies that your program is
using STOKEN to identify a hiperspace as the data object. The hiperspace
must be a nonshared standard type and must be owned by the task issuing the
IDENTIFY. Only the owner of the hiperspace can issue any subsequent
ACCESS, MAP, and SAVE. You can use a nonshared standard hiperspace if no
program has ever issued ALESERV ADD for that hiperspace. You cannot issue
ALESERV ADD for a nonshared standard hiperspace while it is a data object.

,PFCOUNT=nnn

Specifies the additional pages the system is to read into real storage on a page
fault. nnn is an unsigned decimal number from 0 to 255. If you specify an
integer greater than 255, the system uses 255. Zero is the default. If you omit
PFCOUNT or specify PFCOUNT=0, the system reads blocks from the data
object one at a time. In any case, the system reads in successive pages only to
the end of the virtual range of the mapped area containing the originally
referenced page.

Use PFCOUNT if your program accesses the mapped object in a sequential
manner. Because you get a page fault the first time you reference each page,
reading into real storage multiple consecutive pages on each page fault might
decrease the number of page faults and improve your program's performance.

PFCOUNT applies to movement of pages from the object to central storage.
PFCOUNT does not apply to movement of changed or unchanged data that the
system has moved to the real storage as a direct result of system management
of the real storage.

,RELEASE=YES
,RELEASE=NO

Specify RELEASE=YES to release all virtual pages in the reset range. Specify
RELEASE=NO or use the default to release only changed pages in the reset
range. RELEASE=NO does not replace unchanged pages in the window with a
new copy of pages from the object. It replaces only changed pages. If another
ID might have changed the object itself while you viewed data in the window,
specify RELEASE=YES to reset all pages. Any subsequent reference to these
pages will cause the system to load a new copy of the data page from the
object.

ABEND codes

DIV might abnormally terminate with abend code X'08B'. See z/OS MVS System
Codes for an explanation and programmer response.
DIV macro

Return and reason codes

When the system returns control to the caller after the DIV macro executes, it supplies a return code in the low-order (rightmost) byte of general register 15 and a reason code in the two low-order bytes of register 0. After an unsuccessful completion, the system abnormally terminates the caller and supplies an abend code of X'08B' and a reason code in the two low-order bytes of general register 15. See [z/OS MVS System Codes](#) for a detailed explanation of the reason codes for abend code X'08B'.

The hexadecimal values of the reason and return codes are:

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| none        | 00          | —          | **Meaning**: Successful completion.  
              |             |            | **Action**: None.             |
| 0001        | none        | 08B        | **Meaning**: Unknown service was requested.  
              |             |            | **Action**: None.             |
| 0002        | none        | 08B        | **Meaning**: Unknown parameter list format.  
              |             |            | **Action**: None.             |
| 0003        | none        | 08B        | **Meaning**: Input parameter list cannot be addressed.  
              |             |            | **Action**: None.             |
| 0004        | none        | 08B        | **Meaning**: Storage specified in the parameter list cannot be addressed.  
              |             |            | **Action**: None.             |
| 0005        | none        | 08B        | **Meaning**: The parameter list contains a reserved field that does not contain binary zeros.  
              |             |            | **Action**: None.             |
| 0006        | none        | 08B        | **Meaning**: The caller is not running in task mode.  
              |             |            | **Action**: None.             |
| 0007        | none        | 08B        | **Meaning**: The caller is in cross memory mode.  
              |             |            | **Action**: None.             |
| 0008        | none        | 08B        | **Meaning**: The specified TYPE is not valid.  
              |             |            | **Action**: None.             |
| 0009        | none        | 08B        | **Meaning**: The supplied ID is not valid or is an ID that the caller cannot use.  
              |             |            | **Action**: None.             |
| 000A        | 08          | —          | **Meaning**: Environmental error. Another service is currently executing with the specified ID.  
<pre><code>          |             |            | **Action**: Retry the request one or more times until the other service currently executing for this ID completes. |
</code></pre>
<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 000B        | none        | 08B        | **Meaning:** The object is already accessed with the specified ID.  
**Action:** None. |
| 000C        | none        | 08B        | **Meaning:** The caller does not have proper RACF® authorization to the requested object.  
**Action:** None. |
| 000D        | none        | 08B        | **Meaning:** The requested window exceeds the maximum allowable size for the object.  
**Action:** None. |
| 000E        | none        | 08B        | **Meaning:** The object is not currently accessed for the specified ID.  
**Action:** None. |
| 000F        | none        | 08B        | **Meaning:** The specified range overlaps a range that is already mapped for the specified ID.  
**Action:** None. |
| 0010        | none        | 08B        | **Meaning:** The specified range overlaps another mapped range in the current address space or in the specified data space.  
**Action:** None. |
| 0011        | none        | 08B        | **Meaning:** Undetermined user error.  
**Action:** None. |
| 0012        | none        | 08B        | **Meaning:** The virtual storage specified does not begin on a 4K boundary.  
**Action:** None. |
| 0013        | none        | 08B        | **Meaning:** The virtual storage specified is not in a pageable private area subpool.  
**Action:** None. |
| 0014        | none        | 08B        | **Meaning:** The virtual range specified cannot be used to map an object.  
**Action:** None. |
| 0015        | none        | 08B        | **Meaning:** The caller did not issue GETMAIN for at least one page in the specified range.  
**Action:** None. |
| 0016        | none        | 08B        | **Meaning:** The virtual range specified contains at least one fixed page and you did not specify RETAIN=YES.  
**Action:** None. |
### Reason Code Return Code Abend Code Meaning and Action

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0017</td>
<td>0C</td>
<td>—</td>
<td><strong>Meaning:</strong> System error. Portions of virtual storage mapping the object were not addressable, and therefore, could not be saved. (There was either a paging I/O error or data occupying a bad real frame.)&lt;br&gt;<strong>Action:</strong> Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel.</td>
<td></td>
</tr>
<tr>
<td>0018</td>
<td>none</td>
<td>08B</td>
<td><strong>Meaning:</strong> The caller does not have UPDATE access to the object.&lt;br&gt;<strong>Action:</strong> None.</td>
<td></td>
</tr>
<tr>
<td>0019</td>
<td>none</td>
<td>08B</td>
<td><strong>Meaning:</strong> A page to be saved or reset was in the page fixed state.&lt;br&gt;<strong>Action:</strong> None.</td>
<td></td>
</tr>
<tr>
<td>001A</td>
<td>04</td>
<td>—</td>
<td><strong>Meaning:</strong> Program error. The specified range does not encompass any mapped area of the object.&lt;br&gt;<strong>Action:</strong> None required. However, you might want to check that the specified range for this operation was correct.</td>
<td></td>
</tr>
<tr>
<td>001B</td>
<td>none</td>
<td>08B</td>
<td><strong>Meaning:</strong> The virtual storage area specified to be unmapped is not currently mapped.&lt;br&gt;<strong>Action:</strong> None.</td>
<td></td>
</tr>
<tr>
<td>001C</td>
<td>08</td>
<td>—</td>
<td><strong>Meaning:</strong> Environmental error. The object cannot be accessed at the current time.&lt;br&gt;<strong>Action:</strong> Retry the request one or more times until the operation succeeds.</td>
<td></td>
</tr>
<tr>
<td>001D</td>
<td>none</td>
<td>08B</td>
<td><strong>Meaning:</strong> The accessed object is not at the correct control interval size.&lt;br&gt;<strong>Action:</strong> None.</td>
<td></td>
</tr>
<tr>
<td>001E</td>
<td>none</td>
<td>08B</td>
<td><strong>Meaning:</strong> The length of the dname exceeds the maximum size allowed.&lt;br&gt;<strong>Action:</strong> None.</td>
<td></td>
</tr>
<tr>
<td>001F</td>
<td>none</td>
<td>08B</td>
<td><strong>Meaning:</strong> The caller's storage protect key is not the same as when IDENTIFY was invoked.&lt;br&gt;<strong>Action:</strong> None.</td>
<td></td>
</tr>
<tr>
<td>0020</td>
<td>none</td>
<td>08B</td>
<td><strong>Meaning:</strong> An ACCESS was attempted by a task that does not own the specified ID.&lt;br&gt;<strong>Action:</strong> None.</td>
<td></td>
</tr>
</tbody>
</table>
## DIV macro

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0021        | 0C          | —          | **Meaning:** System error. Portions of the object could not be retained in virtual storage as requested.  
**Action:** Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel. |
| 0022        | none        | 08B        | **Meaning:** The task that issued IDENTIFY (or the task for which it is a subtask) does not own the virtual storage it is attempting to map.  
**Action:** None. |
| 0023        | none        | 08B        | **Meaning:** Part or all of the specified storage to be mapped is not in the user's key.  
**Action:** None. |
| 0024        | none        | 08B        | **Meaning:** The caller requested a DIV service holding the local lock.  
**Action:** None. |
| 0025        | none        | 08B        | **Meaning:** The caller requested a DIV service while not in a correct calling environment.  
**Action:** None. |
| 0026        | none        | 08B        | **Meaning:** The caller requested a DIV service, but was not in a 31-bit addressing mode.  
**Action:** None. |
| 0027        | none        | 08B        | **Meaning:** The specified offset and span describe a range that goes beyond the maximum supported object size.  
**Action:** None. |
| 0028        | 08          | —          | **Meaning:** Program error. The caller tried to access an empty data set with MODE=READ specified.  
**Action:** None required. If the data set was not expected to be empty, check return codes from previous DIV operations to ensure that the data was saved as expected. |
| 0029        | none        | 08B        | **Meaning:** The caller tried to map into a disabled reference (DREF) data space.  
**Action:** None. |
| 002A        | none        | 08B        | **Meaning:** The caller tried to map the object into a data space. However, the caller has specified LOCVIEW=MAP to access the object.  
**Action:** None. |
### DIV macro

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 002B        | none        | 08B        | **Meaning:** The data space is not big enough to contain the window. 
**Action:** None. |
| 002C        | none        | 08B        | **Meaning:** The caller requested a data space or hiperspace MAP with address space MAPs outstanding, or an address space MAP with data space or hiperspace MAPs outstanding under the given ID. 
**Action:** None. |
| 002D        | 04          | —          | **Meaning:** The data space has been deleted. However, the requested UNMAP has been successful. 
**Action:** None. |
| 002E        | none        | 08B        | **Meaning:** The data space has been deleted. The requested UNMAP cannot be performed. At least one page in the SAVELIST range was in a deleted data space. 
**Action:** None. |
| 0036        | none        | 08B        | **Meaning:** STOKEN does not represent a valid data space that the caller can use. 
**Action:** None. |
| 0037        | 04          | —          | **Meaning:** Program error. The caller invoked ACCESS. The ACCESS is successful, but the system is issuing a warning that the data set was not allocated with a SHAREOPTIONS(1,3) and that LOCVIEW=MAP was not specified with ACCESS. 
**Action:** None required. However, to eliminate the possibility of potential errors, you should allocate the data set to be used as a DIV object with SHAREOPTIONS(1,3), or you should specify LOCVIEW=MAP when the DIV ACCESS is done. |
| 0038        | none        | 08B        | **Meaning:** The caller invoked ACCESS, but ACCESS failed because the data set was not allocated as a linear data set. 
**Action:** None. |
| 0039        | none        | 08B        | **Meaning:** The caller specified SAVE or RESET for a storage range that contains DREF storage. The SAVE or RESET was unsuccessful. 
**Action:** None. |
| 003A        | none        | 08B        | **Meaning:** The program attempted to map an ESO hiperspace. You can map only to a standard type hiperspace. 
**Action:** None. |
<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 003B        | none        | 08B        | **Meaning:** The caller requested UNMAP with RETAIN=YES for a hiperspace window. You must specify RETAIN=NO or use the default.  
**Action:** None. |
| 003C        | none        | 08B        | **Meaning:** The caller requested UNMAP with RETAIN=YES for a mapped standard hiperspace object. You must specify RETAIN=NO or use the default.  
**Action:** None. |
| 003D        | none        | 08B        | **Meaning:** The STOKEN for the object associated with the specified ID does not represent a valid hiperspace that this request can use.  
**Action:** None. |
| 003E        | 08          | —          | **Meaning:** Environmental error. The hiperspace object cannot be accessed at this time. The number of current READs might exceed the maximum allowed. (If MODE=READ, the object is already accessed under a different ID for UPDATE. If MODE=UPDATE, the object is already accessed under at least one other ID.)  
**Action:** Retry the request one or more times until the operation succeeds. |
| 003F        | none        | 08B        | **Meaning:** The caller specified LOCVIEW=MAP for an ID associated with a hiperspace object.  
**Action:** None. |
| 0040        | 08          | —          | **Meaning:** Environmental error. The specified MAP range would extend the data object beyond the installation data space limits.  
**Action:** Retry the MAP operation with a smaller range specified, or map this range onto a different DIV object. |
| 0041        | none        | 08B        | **Meaning:** The caller specified a STOKEN with an ID representing a hiperspace object. Mapping data space virtual storage onto a hiperspace object is not allowed.  
**Action:** None. |
| 0042        | none        | 08B        | **Meaning:** The hiperspace you are specifying as a data object has been the object of an ALESERV ADD macro, and is therefore ineligible to be used as a DIV object.  
**Action:** None. |
<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0043        | 04          | —          | **Meaning:** Program error. The specified range has no pages that have been altered.  
**Action:** None required. However, you might want to check that the specified range for this operation was correct. |
| 0044        | 04          | —          | **Meaning:** Successful completion. The table is full and there are more ranges to check.  
**Action:** None required. However, to obtain all of the information regarding changed pages, you can either retry the SAVELIST operation with a larger list, or you can obtain a new OFFSET and SPAN from the last entry in the returned list, and invoke SAVELIST another time to fill in the list with additional changed page information. |
| 0045        | 08          | —          | **Meaning:** Environmental error. Storage for the SAVELIST operation could not be obtained. The DIV request is rejected.  
**Action:** Retry the request 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. |
| 0046        | none        | 08B        | **Meaning:** The LISTSIZE specified is not valid.  
**Action:** None. |
| 0047        | none        | 08B        | **Meaning:** SAVE and either LISTADDR or LISTSIZE is specified.  
**Action:** None. |
| 0048        | none        | 08B        | **Meaning:** All or a portion of a range specified in the user's SAVELIST does not intersect with a mapped region.  
**Action:** None. |
| 0049        | none        | 08B        | **Meaning:** While using a user list with SAVE, the caller specified either OFFSET or SPAN.  
**Action:** None. |
| 004A        | none        | 08B        | **Meaning:** Addresses in the user list are not valid, not on a page boundary, or the start address is higher than the end address.  
**Action:** None. |
| 004B        | none        | 08B        | **Meaning:** Selected range spans across multiple data spaces or hiperspaces.  
**Action:** None. |
### DIV macro

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 004C        | none        | 08B        | **Meaning:** The caller specified SAVE for a data space or hiperspace, but did not supply a value for STOKEN.  
**Action:** None. |
| 004D        | none        | 08B        | **Meaning:** The caller is not authorized to access the requested data.  
**Action:** None. |
| 0052        | none        | 08B        | **Meaning:** The specified virtual range contains at least one protected page.  
**Action:** Remove the protection status from the protected pages in the specified virtual range. Then issue the DIV macro again. If you want to invoke MAP or UNMAP and want to preserve the protection status, specify RETAIN=YES when you issue the macro. |
| 0801        | 08          | —          | **Meaning:** Environmental error. Storage to build the necessary data-in-virtual control block structure could not be obtained.  
**Action:** Retry the request 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. |
| 0802        | 08          | —          | **Meaning:** System error. I/O driver failure.  
**Action:** Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel. |
| 0803        | 0C          | —          | **Meaning:** System error. A necessary page table could not be read into central (also called real) storage.  
**Action:** Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel. |
| 0804        | 0C          | —          | **Meaning:** System error. Catalog update failed.  
**Action:** Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel. |
| 0805        | none        | 08B        | **Meaning:** System error. Indeterminate origin.  
**Action:** None. |
### DIV macro

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Return Code</th>
<th>Abend Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0806        | 0C          | —          | **Meaning:** System error. I/O error.  
**Action:** Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel. |
| 0807        | 04          | —          | **Meaning:** Environmental error. Media damage might be present in allocated DASD space. The damage is beyond the currently saved portion of the object. The SAVE completed successfully.  
**Action:** None required. However, do not attempt to increase the size of this DIV object. |
| 0808        | 08          | —          | **Meaning:** System error. I/O from a previous request has not completed.  
**Action:** Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel. |

### Example 1

Identify a hiperspace as a data object. The hiperspace's STOKEN is at HSSTOK. IDENTIFY is to return the ID at DIVOBJID.

```
DIV IDENTIFY,TYPE=HS,STOKEN=HSSTOK,ID=DIVOBJID
```

### Example 2

Whenever a page fault on a page in the mapped range requires that the system read the page from the data set object, the system, if possible, preloads up to seven additional pages, virtually successive to the fault page.

```
DIV MAP,ID=DIVOBJID,AREA=MAPPTR1,SPAN=SPANVAL,OFFSET=*,STOKEN=DSSTOK,PFCOUNT=7
```

### DIV-List form

#### Syntax

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

```
name name

/bslash

DIV

/bslash

name: Symbol. Begin name in column 1.

b

One or more blanks must precede DIV.

DIV

One or more blanks must follow DIV.
```
DIV macro

Valid parameters:
(Underlined parameters are those that you must specify.)

IDENTIFY
ID, TYPE, DDNAME or STOKEN

ACCESS
ID, MODE, SIZE, LOCVIEW

MAP
ID, AREA, OFFSET, SPAN, STOKEN, RETAIN, PFCOUNT

RESET
ID, OFFSET, SPAN, RELEASE

SAVE
ID, OFFSET, SPAN, SIZE, LISTADDR, LISTSIZE, STOKEN

SAVELIST
ID, LISTADDR, LISTADDR, STOKEN

UNMAP
ID, AREA, STOKEN

UNACCESS
ID

UNIDENTIFY
ID

, ID = addr
addr: A-type address

, AREA = addr
addr: A-type address

, DDNAME = addr
addr: A-type address

, LISTADDR = addr
addr: RX-type address, or register (2) - (12).

, LISTSIZE = addr
addr: RX-type address, or register (2) - (12).

, LOCVIEW = MAP
Default: LOCVIEW = NONE

, LOCVIEW = NONE

, MODE = READ
Default: None

, MODE = UPDATE

, OFFSET = addr
addr: A-type address

, OFFSET = *

, RETAIN = YES
Default: RETAIN = NO

, RETAIN = NO

, SIZE = addr
addr: A-type address

, SIZE = *

, SPAN = addr
addr: A-type address

, SPAN = *

, STOKEN = addr
addr: A-type address

, TYPE = DA
Default: None

, TYPE = HS

, PFCOUNT = nnn
Default: 0

, RELEASE = YES
Default: RELEASE = NO

, RELEASE = NO

, MF = L
See explanation of parameters if omitted.
DIV macro

Parameters

\^,MF=L

Specifies the list form of the DIV macro. The list form generates the DIV parameter list in line without any executable code or register usage.

DIV-Execute form

Syntax

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

```
name name

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

b

One or more blanks must precede DIV.

DIV

b

One or more blanks must follow DIV.

<table>
<thead>
<tr>
<th>Valid parameters:</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Underlined parameters are those that you must specify.)</td>
</tr>
</tbody>
</table>

- **IDENTIFY**
  - ID, TYPE, DDNAME or STOKEN
- **ACCESS**
  - ID, MODE, SIZE, LOCVIEW
- **MAP**
  - ID, AREA, OFFSET, SPAN, STOKEN, RETAIN, PFCOUNT
- **RESET**
  - ID, OFFSET, SPAN, RELEASE
- **SAVE**
  - ID, OFFSET, SPAN, SIZE, STOKEN, LISTADDR, LISTSIZE, MF
- **SAVELIST**
  - ID, LISTADDR, LISTSIZE, MF
- **UNMAP**
  - ID, AREA, RETAIN, STOKEN
- **UNACCESS**
  - ID
- **UNIDENTIFY**
  - ID

\^ID=addr

addr: RX-type address, or register (2) - (12).

\^AREA=addr

addr: RX-type address, or register (2) - (12).

\^DDNAME=addr

addr: RX-type address, or register (2) - (12).

\^LISTADDR=addr

addr: RX-type address, or register (2) - (12).

\^LISTSIZE=addr

addr: RX-type address, or register (2) - (12).

\^LOCVIEW=MAP

\^LOCVIEW=NONE

Default: LOCVIEW=NONE

\^MODE=READ

\^MODE=UPDATE

Default: None

\^OFFSET=addr

\^OFFSET=*

addr: RX-type address, or register (2) - (12).

\^RETAIN=YES

Default: RETAIN=NO.
DIV macro

,RETAIN=NO

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

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

,STOKEN=addr
  addr: RX-type address.

>Type=DA
Default: None

,TYPE=HS

,PFCOUNT=nnn
Default: 0

,RELEASE=YES
Default: RELEASE=NO

,RELEASE=NO

,MF=(E,addr)

Parameters

,MF=(E,addr)
Specifies the execute form. In the execute form, DIV will be called using the parameter list specified by “addr”. “addr” indicates the address of the parameter list and may be (a) any address that is valid in an RX-type assembler language instruction or (b) one of the general registers 2 through 12 specified within parentheses. The register may be expressed either symbolically or as a decimal integer. The specified parameter list will be updated for any parameters that are specified. Other parameter fields will be unaffected.

DIV—Modify form

Syntax

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

\( name \)  
name: Symbol. Begin name in column 1.

b
One or more blanks must precede DIV.

DIV

b
One or more blanks must follow DIV.

Valid parameters:
(Underlined parameters are those that you must specify.)

IDENTIFY
ID, TYPE, DDNAME or STOKEN
DIV macro

ACCESS
MAP
RESET
SAVE
SAVELIST
UNMAP
UNACCESS
UNIDENTIFY

[ID=addr]

,AREA=addr
,DDNAME=addr
,LISTADDR=addr
,LISTSIZE=addr
,LOCVIEW=MAP

,LOCVIEW=NONE
,MODE=READ
,MODE=UPDATE

,OFFSET=addr
,OFFSET=* 
,RETAIN=YES
,RETAIN=NO

,SIZE=addr
,SIZE=* 
,SPAN=addr
,SPAN=* 
,STOKEN=addr

,TYPE=DA
,TYPE=HS

,PFCOUNT=nnn
,RELEASE=YES
,RELEASE=NO

,MF=(M,addr) See explanation of parameters if omitted.
Parameters

\( M, \text{addr} \)

Specifies the MODIFY form. The modify form of the macro is used to modify an already defined DIV parameter list. It is exactly the same as the EXECUTE form except that DIV is not called. The contents of registers 1 and 15 are destroyed.
DIV macro
Chapter 89. DOM — Delete operator message

Description

The DOM macro deletes an operator message or group of messages from the display screen of the operator's console. It can also prevent messages from ever appearing on any operator's console. When a program no longer requires that a message be displayed, it can issue the DOM macro to delete the message.

Depending on the timing of the DOM relative to the WTO(R), the message may or may not be displayed. If the message is being displayed, it is removed when space is required for other messages.

When a WTO or WTOR macro is issued, the system assigns an identification number to the message and returns this number (24 bits right-justified) to the issuing program in general register 1. When you no longer need this message displayed, issue the DOM macro using the identification number that was returned in general register 1.

Environment

The requirements for the caller are:

- Minimum authorization: Problem state and any PSW key.
- Dispatchable unit mode: Task
- Cross memory mode: PASN=HASN=SASN
- AMODE: 24- or 31- or 64-bit
- ASC mode: Primary
- Interrupt status: Enabled for I/O and external interrupts
- Locks: No locks held
- Control parameters: Must be in the primary address space.

Programming requirements

None.

Restrictions

If you are deleting messages by lists of DOM IDs, you cannot delete more than 60 at a time.

Register information

Input register information

Before issuing the DOM 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-14</td>
<td>Unchanged</td>
</tr>
<tr>
<td>15</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>

© Copyright IBM Corp. 1988, 2012
DOM macro

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 DOM macro is written as follows:

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

b
  One or more blanks must precede DOM.

DOM
  One or more blanks must follow DOM.

MSG=addr
  addr: Register (1) - (12), or an address.

MSGLIST=list addr
  list addr: Symbol, RX-type address, or register (1) - (12).

TOKEN=addr
  addr: Register (1) - (12), or an address.

, COUNT=addr
  addr: Register (2) - (12), or an address.
```

Parameters

The parameters are explained as follows:

- **MSG=addr**
  - addr: Register (1) - (12), or an address.

- **MSGLIST=list addr**
  - list addr: Symbol, RX-type address, or register (1) - (12).

- **TOKEN=addr**
  - addr: Register (1) - (12), or an address.

- **, COUNT=addr**
  - addr: Register (2) - (12), or an address.

For MSG, the address or register contains the 32-bit, right-justified identification number of the message to be deleted. Use this parameter to delete a single message.

For MSGLIST, the address is of a list of one or more fullwords, each word containing a 32-bit, right-justified identification number of a message to be deleted. A maximum of 60 identification numbers may be in the message list. If more than 60 identification numbers are in the list, only the first 60 are processed. Begin the list on a fullword boundary. When you are not using the COUNT parameter, indicate the end of the list by setting the high-order bit of the last fullword entry to 1.

Attention: DOM ids should not be altered from the 32-bit value returned in register 1 by the WTO or WTOR macro, except to turn on the high-order bit (x'80000000') in the last entry in a list.
Specifies a field or register containing a 4-byte token that is associated with messages to be deleted. Using the TOKEN parameter is an alternate method for identifying messages, which is independent of the register 1 message ID. When you issue WTO or WTOR to write a message, you can specify a token value. To delete that WTO or WTOR message, specify the same token value by issuing DOM with the TOKEN parameter. You cannot use the token value on the DOM macro unless you specified that token value on the WTO or WTOR macro that wrote the message. Issuing DOM with the TOKEN parameter deletes all messages issued through WTO or WTOR with the same token value. Unauthorized users may delete only those messages which were originally issued under the same jobstep TCB, ASID and system ID. The value of the token may not be the same as the ID that was returned in register 1 after a WTO or WTOR. This keyword is mutually exclusive with the MSG, MSGLIST, and COUNT keywords.

The count field or register contains the one-byte count of messages to be deleted (specified on the MSG or MSGLIST parameters) associated with this request. The count value must be from 1 to 60. If this keyword is used, the issuer must not set the high order bit on in the last entry of the DOM parameter list. If this keyword is not specified, the DOM ids are treated as 32-bit ids. If an address is used instead of a register, the address points to a 1-byte field which contains the count. The COUNT keyword is invalid with the TOKEN keyword.

Note: For any DOM keywords that allow a register specification, the value must be right-justified in the register and the remaining bytes within the register must be zero.

Example 1

Delete an operator message whose message id is in register 1.

```
DOM MSG=(1)
```

Example 2

Delete a number of operator messages. The COUNT parameter indicates how many messages are to be deleted.

```
DOM MSGLIST=ID3,COUNT=COUNT4
```

Example 3

Delete all messages issued with a particular token.

```
DOM TOKEN=TOKEN1
```
DOM macro


Chapter 90. DSPSERV — Create, delete, and control data spaces

Description

DSPSERV for hiperspaces

To control the use of hiperspaces, use the variation of the DSPSERV macro described under Chapter 91, “DSPSERV — Create, delete, and control hiperspaces,” on page 505.

The DSPSERV macro creates, deletes, and controls data spaces. A data space is a range of up to two gigabytes of contiguous virtual storage addresses that a program can directly manipulate through assembler instructions. A data space can hold only user data and user programs stored as data; code does not execute in a data space.

There are three kinds of data spaces: SCOPE=SINGLE, SCOPE=ALL, and SCOPE=COMMON. A SCOPE=SINGLE data space is used in ways similar to the use of the private area of an address space. A SCOPE=ALL or SCOPE=COMMON data space is used in ways similar to the use of the common area of an address space. A problem state program with PSW key 8-F cannot create or delete a SCOPE=ALL or SCOPE=COMMON data space. However, it can use these spaces, providing a supervisor state program or a program with PSW key 0-7 created the space and established addressability to the space on its behalf. For more information on data spaces and how to use them, see z/OS MVS Programming: Assembler Services Guide.

Use the DSPSERV macro to:

- Create a data space (CREATE parameter and TYPE=BASIC parameter)
- Delete a data space (DELETE parameter)
- Release an area of a data space (RELEASE parameter)
- Increase the current size of a data space (EXTEND parameter)
- Load an area of a data space into central storage (LOAD parameter)
- Take (that is, page out) from real storage an area of a data space (OUT parameter)

On the DSPSERV macro, data spaces are identified through STOKENs. A STOKEN is a unique identifier of address spaces, data spaces, andhiperspaces.

Environment

The requirements for the caller are:

**Minimum authorization:** A problem state program with PSW key 8-F can use DSPSERV to create a SCOPE=SINGLE data space. For all other DSPSERV services, that program must own the data space.

**Dispatchable unit mode:** Task or SRB

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

**AMODE:** 31- or 64-bit

**ASC mode:** Primary or access register (AR)

**Interrupt status:** Enabled for I/O and external interrupts

**Locks:** No locks held

**Control parameters:** Must be in the primary address space.
DSPSERV macro for data spaces

Programming requirements
If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before you issue DSPSERV. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

If you use the RELEASE parameter to specify a range of storage using INLIST=YES, you must use the RANGLIST parameter to specify a range list that is mapped by the IARDRL macro. For information on the IARDRL macro, see z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/

Restrictions
None.

Input register information
Before issuing the DSPSERV 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 DSPSERV macro is written as follows:

```
name
```

name: Symbol. Begin name in column 1.
DSPSERV macro for data spaces

One or more blanks must precede DSPSERV.

DSPSERV

One or more blanks must follow DSPSERV.

---

Valid parameters (Required parameters are underlined.)

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>STOKEN, NAME, TYPE, GENNAME, OUTNAME, BACK,</td>
</tr>
<tr>
<td></td>
<td>BLOCKS, TTOKEN, ORIGIN, NUMBLKS</td>
</tr>
<tr>
<td>RELEASE</td>
<td>STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE</td>
</tr>
<tr>
<td>DELETE</td>
<td>STOKEN</td>
</tr>
<tr>
<td>EXTEND</td>
<td>STOKEN, BLOCKS, VAR, NUMBLKS</td>
</tr>
<tr>
<td>LOAD</td>
<td>STOKEN, BLOCKS, START</td>
</tr>
<tr>
<td>OUT</td>
<td>STOKEN, BLOCKS, START</td>
</tr>
</tbody>
</table>

,STOKEN=stoken-addr

STOKEN: RX-type address or register (2) - (12).

,TYPE=BASIC

TYPE: BASIC

,NAME=name-addr

NAME: RX-type address or register (2) - (12).

,GENNAME=NO

GENNAME: NO

,GENNAME=COND

Default: GENNAME=NO

,GENNAME=YES

,OUTNAME=outname-addr

OUTNAME: RX-type address or register (2) - (12).

,START=start-addr

START: RX-type address or register (2) - (12).

,BLOCKS=(max-addr,init-addr)

BLOCKS: RX-type address or register (2) - (12).

,BLOCKS=(max)init

init: RX-type address or register (2) - (12).

,BLOCKS=max

max: Number up to 524288.

,BLOCKS=(0,init)

init: Number up to 524288.

,BLOCKS=0

0 specifies the installation default size.

Default for CREATE: BLOCKS=0

,BLOCKS=(size-addr)

size-addr: RX-type address or register (2) - (12).

,BLOCKS=(size)

size: Number up to 524288.

,BACK=31

Default: BACK=31

,BACK=64

,TOKEN=ttoken-addr

ttoken-addr: RX-type address or register (2) - (12).

,ORIGIN=origin-addr

origin-addr: RX-type address or register (2) - (12).

,NUMBLKS=numblks-addr

numblks-addr: RX-type address or register (2) - (12).

,INLIST=NO

Default: INLIST=NO

,INLIST=YES

,RANGLIST=rangelist_addr

rangelist_addr: RS-type address or register (2) - (12). Required with INLIST=YES.

,NUMRANGE=numrange_addr

numrange_addr: RS-type address or register (2) - (12).
**DSPSERV macro for data spaces**

,NUMRANGE=1  
Default: NUMRANGE=1

,VAR=NO  
Default: VAR=NO

,VAR=YES

,PLISTVER=IMPLIED_VERSION  
Default: IMPLIED_VERSION

,PLISTVER=MAX

,PLISTVER=plistver

,MF=S

---

**Parameters**

The CREATE, RELEASE, DELETE, EXTEND, LOAD, and OUT parameters, which designate the services of the DSPSERV macro, are mutually exclusive. You can select only one.

The parameters are explained as follows:

**CREATE**

Requests that the system create a data space. Creating a data space is somewhat like issuing a GETMAIN for storage. The entire data space is in the same storage key. When you specify CREATE, you must specify the NAME and STOKEN parameters.

Optional parameters when you create a data space are: TYPE, GENNAME, OUTNAME, BLOCKS, BACK, TTOKEN, ORIGIN, and NUMBLKS.

**RELEASE**

Requests that the system resources used to contain the user's data be returned to the system. Although the data contained in the virtual storage is discarded, the user's virtual storage itself remains and is available for further use. When you specify RELEASE, you must also specify STOKEN to identify the data space, and the START and BLOCKS parameters to identify the beginning and the length of the area to be returned to the system.

A problem state program with PSW key 8 - F can release any data space it owns or created, providing its PSW key matches the storage key of the data space. Note that no exception to the caller's PSW key being zero or equal to the key of the storage to be released is made for a storage-protection override. However, if the program is using the IARVSERVER macro to share the data space, the program cannot release the data space if it is a shared group and is fixed through another view.

Use DSPSERV RELEASE instead of using the MVCL instruction for these reasons:
- The DSPSERV RELEASE is faster than MVCL for very large areas.
- Pages that are released through DSPSERV RELEASE do not occupy space in real storage.

**DELETE**

Requests that the system delete a data space. STOKEN is the only required parameter on the DELETE request.

A problem state program with PSW key 8 - F can delete any data space it owns or created, providing its PSW key matches the storage key of the data space.
DSPSERV macro for data spaces

EXTEND
Requests that the system increase the current size of a data space. Use EXTEND only for a data space that was created with an initial size smaller than a maximum size. Before a caller can reference storage beyond the current size, the caller must use EXTEND to increase the storage that is available. If a caller references hiperspace storage beyond the current size, the system rejects the request; it terminates the caller with an 0C4 abend code.

STOKEN (identifying the data space) and BLOCKS (specifying the size of the increase) are required on the EXTEND request. VAR (requesting a variable extension) and NUMBLKS (requesting the size of the extension) are optional parameters.

For a problem state and PSW key 8 - F caller, any TCB can extend a data space that was created by any other TCB in the address space.

The system rejects the EXTEND request if you specified VAR=NO (or took the default) and the extended size would:
- Exceed the maximum size specified when the data space was created.
- For a data space with a storage key greater than 7, extend the cumulative data space and hiperspace space totals beyond the installation limits for the owning address space.

LOAD
Requests that the system load some areas of a data space into central storage. The system fills the request depending on how many central storage frames are available. When you specify LOAD, you must also specify the STOKEN, START, and BLOCKS parameters.

For a problem state and PSW key 8 - F caller, the TCB that represents it owns the data space.

OUT
Tells the system that it can take some areas of a data space from central storage. When you specify OUT, you must also specify the STOKEN, START, and BLOCKS parameters.

For a problem state and PSW key 8 - F caller, the TCB that represents it owns the data space.

\,STOKEN=stoken-addr
Specifies the address of the eight-byte STOKEN for the data space. DSPSERV CREATE returns the STOKEN as output; STOKEN is required input for all other DSPSERV requests.

\,TYPE=BASIC
Specifies that the system should create a data space rather than a hiperspace. TYPE=BASIC is the default.

\,NAME=name-addr
Specifies the address of the eight-byte variable or constant that contains the name of the data space. NAME is required for DSPSERV CREATE.

Data space names are from one to eight bytes long. They can contain letters, numbers, and @, #, and $, but they cannot contain embedded blanks. Names that contain fewer than eight bytes must be left-justified and padded on the right with blanks.

Data space and hiperspace names must be unique within the home address space of the owner. No other data space or hiperspace in the home address space can have the same name.
DSPSERV macro for data spaces

may have the same name. Therefore, in choosing names for your data spaces, you *must* avoid using the same names that IBM uses for data spaces. Do not use the following names:

- Names that begin with A through I.
- Names that begin with numerals or with SYS.

**How to choose names for your data spaces:**

Use data space names that begin with @, #, $, or the letters J through Z, with the exception of SYS. The system abends problem state programs that begin names with SYS. Do not specify a data space name beginning with a numeric if you are creating the data space name.

To ensure that the names for your data spaces are unique, use the GENNAME parameter to generate a unique name.

\[ \text{GENNAME} = \text{NO} \]
\[ \text{GENNAME} = \text{COND} \]
\[ \text{GENNAME} = \text{YES} \]

Specifies whether or not you want the system to generate a name for the data space to ensure that all names are unique within the address space. The system generates a name by adding a 5-character prefix (consisting of a numeral followed by four characters) to the first three characters of the name you supply on the NAME parameter. For example, if you supply 'XYZDATA' on the NAME parameter, the name becomes 'nCCCCXYZ' where 'n' is the numeral, 'CCCC' is the 4-character string generated by the system, and XYZ comes from the name you supplied on NAME. See the NAME parameter for the data space and hiperspace naming conventions.

\[ \text{GENNAME} = \text{NO} \]

The system does not generate a name. You *must* supply a name unique within the address space. GENNAME=NO is the default.

\[ \text{GENNAME} = \text{COND} \]

The system generates a unique name only if you supply a name that is already being used. Otherwise, the system uses the name you supply.

\[ \text{GENNAME} = \text{YES} \]

The system takes the name you supply on the NAME parameter and makes it unique.

If you want the system to return the unique name it generates, use the OUTNAME parameter.

\[ \text{OUTNAME} = \text{OUTNAME-ADDR} \]

Specifies the address of the eight-byte variable where the system returns the name it generated if you specify GENNAME=YES or GENNAME=COND on DSPSERV CREATE. The OUTNAME parameter is optional.

\[ \text{START} = \text{START-ADDR} \]

Specifies the address of a four-byte variable containing the beginning address of a block of storage in a data space. The address must be on a four-kilobyte boundary. START is required on RELEASE requests.

\[ \text{BLOCKS} = (\text{MAX-ADDR}, \text{INIT-ADDR}) \]
\[ \text{BLOCKS} = (\text{MAX}, \text{INIT}) \]
\[ \text{BLOCKS} = \text{MAX} \]
\[ \text{BLOCKS} = (0, \text{INIT}) \]
\[ \text{BLOCKS} = 0 \]
\[ \text{BLOCKS} = (0, \text{INIT-ADDR}) \]
\[ \text{BLOCKS} = \text{SIZE-ADDR} \]
**DSPSERV macro for data spaces**

\[ \text{BLOCKS}=\text{size} \]

Specifies the size of the data space the system is to create, or the size of an area within a data space. BLOCKS is required for all DSPSERV requests except DSPSERV DELETE.

**For a CREATE request**, specifies the maximum size (in blocks) to which the data space can expand (\textit{max-addr} or \textit{max}) and the initial size of the data space (\textit{init-addr} or \textit{init}). A block is a unit of 4K bytes. You cannot extend the data space beyond its maximum size.

\textit{max-addr} specifies the address of a field that contains the maximum size of the data space to be created. \textit{max} is the number of blocks (up to 524,288) to be used for the data space.

\textit{init-addr} specifies the address of the initial size of the data space. \textit{init} is the number of blocks to be used as the initial size. If the initial size you specify exceeds or equals the maximum size, then the initial size becomes the maximum size.

\textit{0} specifies the default size, either the installation default or the IBM-defined default. The IBM-defined default maximum is 239 blocks. Your installation can use the installation exit IEFUSI to change the IBM default. The system returns the maximum size at the location identified by NUMBLKS.

If you do not code the BLOCKS parameter on the CREATE request, the system uses BLOCKS=0, setting the initial size and the maximum size equal to the installation (or IBM) default.

**For a RELEASE request**, BLOCKS and START are required parameters that define contiguous storage (in 4K blocks) that the system is to release. BLOCKS specifies the size of an area to be released (\textit{size-addr} or \textit{size}). The minimum size is 1 block and the maximum is 524,288 blocks (2 gigabytes).

**For an EXTEND request**, BLOCKS is a required parameter that defines the amount of increase of the current size of the data space.

**For LOAD and OUT requests**, BLOCKS is a required parameter that defines the amount of data space storage that the system is to load into central storage or page out from central storage.

\textit{BLOCKS}=\textit{size-addr} in MVS/SP3.1.0 is incompatible with \textit{BLOCKS}=(\textit{size-addr}) in MVS/SP3.1.0e and later releases in the case where \textit{size-addr} is a register. If you coded \textit{BLOCKS}=(\textit{register}) in MVS/SP3.1.0, and plan to recompile the program to run on later releases of MVS, you must change the specification to \textit{BLOCKS}=(\textit{(register)}) before you recompile.

\[ \text{BACK}=31 \]
\[ \text{BACK}=64 \]

Specifies whether the data space pages can be backed by real storage above 2 gigabytes when defined IOON (fixed).

\text{BACK}=31 specifies that the data space pages will be backed by frames below 2 gigabytes when defined IOON.

\text{BACK}=64 specifies that the data space pages will be backed by frames above or below 2 gigabytes when defined IOON.

\[ \text{TTOKEN}=\text{ttoken-addr} \]

Specifies the address of the TTOKEN, the 16-byte variable or constant that identifies the task that is to become the owner of the data space. The TTOKEN must represent either the caller's task or the caller's job step task. TTOKEN is valid only on the CREATE request.
DSPSERV macro for data spaces

\[\text{,ORIGIN=origin-addr}\]
Specifies the address of the four-byte variable that contains the lowest address (either zero or 4096) of the new data space. The system returns the beginning address of the data space at origin-addr. The system tries to start all data spaces at origin zero; on some processors, however, the origin is 4096. ORIGIN is an optional parameter for DSPSERV CREATE.

\[\text{,NUMBLKS=numblks-addr}\]
Specifies the address of the four-byte area where the system returns one of the following:
- For DSPSERV CREATE, the maximum size (in blocks) of the newly-created data space.
- For DSPSERV EXTEND, the size by which the system extended the data space.

The NUMBLKS parameter is an optional parameter on DSPSERV CREATE and DSPSERV EXTEND.

If, when you create a data space, you specify BLOCKS=0 or do not specify the BLOCKS parameter, the system uses the default that your installation established in the installation exit IEFUSI. The system returns this default value at numblks-addr.

\[\text{,INLIST=NO}\]
\[\text{,INLIST=YES}\]
Specifies whether a range is included (YES). The default is INLIST=NO. If you specify YES, you must also specify the RANGLIST parameter.

\[\text{,RANGLIST=rangelist-addr}\]
Specifies the name (RS-type) or address (in register 2-12) of a required input fullword that contains the address of the range list. The range list consists of a number of entries (specified by NUMRANGE); each entry is 8 bytes long. A mapping of each entry is provided through the mapping macro IARDRL.

\[\text{,NUMRANGE=numrange_addr}\]
Specifies the name (RS-type) or address (in register 2-12) of an optional parameter that provides the number of entries in the supplied RANGLIST. The maximum value may not exceed 16. The default is 1.

\[\text{,VAR=NO}\]
\[\text{,VAR=YES}\]
Specifies whether or not your request for the system to extend the amount of storage available in a data space is a variable request. When you use DSPSERV EXTEND for a data space, the system might not be able to extend the data space the amount you request because that amount might cause the system to exceed one of the following:
- The maximum size of the data space, as specified on the BLOCKS parameter when the data space was created.
- For a data space with storage key 8 - F, the limit of combined data space and hiperspace storage with storage key 8 - F for an address space. (The installation established this limit on the IEFUSI installation exit, or took the IBM default.)

If you specify VAR=YES (the variable request) and the system cannot satisfy your request, the system extends the data space to one of the following sizes, depending on which is smaller:
- The maximum size specified on the BLOCKS parameter when the data space was created.
DSPSERV macro for data spaces

- The largest size that would still keep the combined data space and hiperspace storage within the limits established by the installation for an address space.

If you specify VAR=NO (the default), the system:
- Abends the caller if the extended size would exceed the maximum size specified when the data space was created.
- Rejects the request if the data space has storage key 8-F and the request would extend the combined data space and hiperspace beyond the installation limit for an address space.

If you use the NUMBLKS parameter, the system returns the size by which the system extends the data space.

\[ \text{,PLISTVER=IMPLIED\_VERSION} \]
\[ \text{,PLISTVER=MAX} \]
\[ \text{,PLISTVER=plistver} \]

Specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:
- **IMPLIED\_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED\_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:
- **IMPLIED\_VERSION**
- **MAX**
- A decimal value of 0

\[ \text{,MF=S} \]

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

ABEND codes

DSPSERV might abnormally terminate with abend code X'01D'. See [z/OS MVS System Codes](#) for an explanation and programmer response.

Return and reason codes

Hexadecimal return and reason codes from DSPSERV CREATE:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>—</td>
<td><strong>Meaning:</strong> DSPSERV CREATE completed successfully. <strong>Action:</strong> None.</td>
</tr>
</tbody>
</table>
DSPSERV macro for data spaces

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04 | 0000Cxx | **Meaning**: Program error. DSPSERV CREATE completed successfully. You specified a size of 2 gigabytes (524,288 blocks). However, because the processor did not support a data space with zero origin, a data space of one less block (524,287 blocks) was created.  
**Action**: None required. However, you should verify that your program correctly accounts for the nonzero origin of the data space. |
| 08 | 00005xx | **Meaning**: Program error. Creation of the data space would violate installation criteria. See the IEFUSI installation exit in [z/OS MVS Installation Exits](https://www.ibm.com/servers/zseries/zos/bkserv/ief/iefus.htm).  
**Action**: Check with your system programmer for local restrictions on the creation of data spaces. |
| 08 | 00009xx | **Meaning**: Program error. The specified data space name is not unique within the address space.  
**Action**: Check that the data space name is not already in use by another active data space. Change the data space name or specify the GENNAME parameter on the DSPSERV macro to get the system to generate a unique name. |
| 08 | 00012xx | **Meaning**: Environmental error. The system's set of generated names for data spaces and hiperspaces has been temporarily depleted.  
**Action**: Retry the job one or more times during a period of lower system usage. If the problem persists, consult your system programmer, who might be able to tune the system so that more names are available for use. |
| 0C | 00006xx | **Meaning**: Environmental error. The system cannot create any additional data spaces at this time because of a shortage of resources.  
**Action**: Retry the job one or more times during a period of lower system usage. If the problem persists, consult your system programmer, who might be able to tune the system so that resources will not become depleted. |
| 0C | 00007xx | **Meaning**: System error. The system cannot obtain addressability to its data structures.  
**Action**: Record the return and reason code and supply it to the appropriate IBM support personnel. |

Hexadecimal return and reason codes from DSPSERV EXTEND:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 00 | — | **Meaning**: DSPSERV EXTEND completed successfully.  
**Action**: None. |
DSPSERV macro for data spaces

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 08          | xx0502xx    | **Meaning:** Environmental error. Extending the data space would cause the data space and hiperspace limits for the address space to be exceeded.  
**Action:** Check with your system programmer, who might be able to tune the system so that the function is made available to your program. |
| 08          | xx0503xx    | **Meaning:** Program error. You are using VAR=YES to extend the current size of the data space; however, the data space is already the maximum size.  
**Action:** None required. However, if your program requires more storage, you should consider creating an additional data space. |

The caller of DSPSERV does not receive any return codes for the RELEASE, DELETE, LOAD, and OUT services.

**Example 1**

Create a data space named TEMP with a size of 10 million bytes.

```
DSP1  DSPSERV CREATE,NAME=DSPCNAME,STOKEN=DSPCSTKN,
      BLOCKS=DSPBLCKS,ORIGIN=DSPCORG

DSPCNAME DC CL8'TEMP'    DATA SPACE NAME
DSPCSTKN DS CL8         DATA SPACE STOKEN
DSPCORG DS F            DATA SPACE ORIGIN RETURNED
DSPCSIZE EQU 10000000   DATA SPACE SIZE
DSPBLCKS DC A((DSPCSIZE+4095)/4096) NUMBER OF BLOCKS NEEDED FOR
                       A 10 MILLION BYTE DATA SPACE
```

**Example 2**

Release a range of storage.

```
LA 5,RANGLST
ST 5,RNGLSTPT
LA 5,RNGLSTPT

DSP2 DSPSERV RELEASE,STOKEN=DSPCSTKN,DISABLED=NO,INLIST=YES,
      NUMRANGE=NUMRANGS,RANGLIST=(5)

RNGLSTPT DS F          RANGE LIST ADDRESS
DSPCSTKN DS CL8        DATA SPACE STOKEN
NUMRANGS DC F'9'       NUMBER OF RANGES TO PROCESS
RANGLST DS CL256       STORAGE FOR MAX NUMBER OF RANGES
DRLMAP DS 0F           THIS CREATES A DSECT
IARDL   OF              MAPPING MACRO
```

DSPSERV—List form

Use the list form of the DSPSERV macro to construct a nonexecutable control program parameter list.

**Syntax**

The list form of the DSPSERV macro is written as follows:
**DSPSERV macro for data spaces**

```
name
```

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

b

One or more blanks must precede DSPSERV.

DSPSERV

b

One or more blanks must follow DSPSERV.

---

```
,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver
```

Default: IMPLIED_VERSION

*plistver*: 0

```
,MF=(L,list addr)
,MF=(L,list addr,attr)
```

*list addr*: Symbol.

*attr*: 1- to 60-character input string.

Default: 0D

---

The parameters are explained as follows:

```
,MF=(L,list addr)
,MF=(L,list addr,attr)
```

Specifies the list form of the DSPSERV macro. *list addr* defines the area that the system is to use for the parameter list.

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

**DSPSERV—Execute form**

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

**Syntax**

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

```
name
```

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

b

One or more blanks must precede DSPSERV.

DSPSERV

b

One or more blanks must follow DSPSERV.
### DSPSERV macro for data spaces

**Valid parameters (Required parameters are underlined.)**

<table>
<thead>
<tr>
<th>Command</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE</td>
<td>STOKEN, NAME, TYPE, GENNAME, OUTNAME, BACK, BLOCKS, TTOKEN, ORIGIN, NUMBLKS</td>
</tr>
<tr>
<td>RELEASE</td>
<td>STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE</td>
</tr>
<tr>
<td>DELETE</td>
<td>STOKEN</td>
</tr>
<tr>
<td>EXTEND</td>
<td>STOKEN, BLOCKS, VAR, NUMBLKS</td>
</tr>
<tr>
<td>LOAD</td>
<td>STOKEN, BLOCKS, START</td>
</tr>
<tr>
<td>OUT</td>
<td>STOKEN, BLOCKS, START</td>
</tr>
</tbody>
</table>

**STOKEN=stoken-addr**

`stoken-addr`: RX-type address or register (2) - (12).

**TYPE=BASIC**

Default: TYPE=BASIC

**NAME=name-addr**

`name-addr`: RX-type address or register (2) - (12).

**GENNAME=NO**

Default: GENNAME=NO

**GENNAME=COND**

**GENNAME=YES**

**OUTNAME=outname-addr**

`outname-addr`: RX-type address or register (2) - (12).

**START=start-addr**

`start-addr`: RX-type address or register (2) - (12).

**BLOCKS=(max-addr,init-addr)**

`max-addr`: RX-type address or register (2) - (12).

**BLOCKS=(max,init)**

`init-addr`: RX-type address or register (2) - (12).

**BLOCKS=max**

`max`: Number up to 524288.

**BLOCKS=0**

0 specifies the installation default size. **Default for CREATE:** BLOCKS=0

**BLOCKS=(0,init-addr)**

**BLOCKS=(size-addr)**

**BLOCKS=(size)**

`size`: Number up to 524288.

**BACK=31**

Default: BACK=31

**BACK=64**

**TTOKEN=ttoken-addr**

`ttoken-addr`: RX-type address or register (2) - (12).

**ORIGIN=origin-addr**

`origin-addr`: RX-type address or register (2) - (12).

**NUMBLKS=numblks-addr**

`numblks-addr`: RX-type address or register (2) - (12).

**INLIST=NO**

Default: INLIST=NO

**INLIST=YES**

**RANGLIST=rangelist-addr**

`rangelist-addr`: RS-type address or register (2) - (12). Required with INLIST=YES

**NUMRANGE=umrang-addr**

`numrange-addr`: RS-type address or register (2) - (12).

Default: NUMRANGE=1

**VAR=NO**

Default: VAR=NO

**VAR=YES**

**PLISTVER=IMPLIED_VERSION**

**PLISTVER=MAX**

**PLISTVER=plistver**

Default: IMPLIED VERSION

`plistver`: 0
DSPSERV macro for data spaces

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

 .MF=(E,list addr)
 .MF=(E,list addr,COMPLETE)

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

 .MF=(E,list addr)
 .MF=(E,list addr,COMPLETE)

  Specifies the execute form of the DSPSERV macro. list addr defines the area
  that the system uses for the parameter list. The system checks for required
  parameters and supplies optional parameters that are not specified.

  COMPLETE specifies that the system is to check for required parameters and
  supply optional parameters that are not specified.
Chapter 91. DSPSERV — Create, delete, and control hiperspaces

Description

DSPSERV for data spaces

To control the use of data spaces, use the variation of the DSPSERV macro described under Chapter 90, "DSPSERV — Create, delete, and control data spaces," on page 491.

The DSPSERV macro creates, deletes, and controls hiperspaces. A hiperspace is a range of up to two gigabytes of contiguous virtual storage addresses that a program can use as a buffer. Like a data space, a hiperspace can hold user data and programs stored as data; it does not contain common areas or system data. Instructions do not execute in a hiperspace. Unlike a data space, data is not directly addressable. To manipulate data in a hiperspace, you bring the data into the address space in 4K-byte blocks.

A nonshared standard hiperspace, available to all programs, is backed with real storage and if necessary, with auxiliary storage. Through the buffer area in the address space, your program can view or "scroll" through the standard hiperspace. A shared standard hiperspace is available to problem state programs with PSW keys of 8 through F, but only under the control of programs in supervisor state or with PSW keys of 0 through 7. An ESO (expanded storage only) hiperspace is available only for supervisor state or PSW key 0 through 7 programs. For more information on hiperspaces and how to use them, see z/OS MVS Programming.

Assembler Services Guide

Use the DSPSERV macro to:

• Create a standard hiperspace (CREATE parameter and TYPE=HIPERSPACE parameter)
• Delete a hiperspace (DELETE parameter)
• Release an area of a hiperspace (RELEASE parameter)
• Increase the current size of a hiperspace (EXTEND parameter)

On the DSPSERV macro, hiperspaces are identified through STOKENs. The STOKEN is a unique identifier of address spaces, data spaces, and hiperspaces.

Environment

The requirements for the caller are:

Minimum authorization: Problem state programs with PSW key 8-F can request these DSPSERV services:

• Create a nonshared standard hiperspace
• Delete any hiperspace they own
• Release an area of a hiperspace
• Increase the current size of a hiperspace

Dispatchable unit mode: Task or SRB
Cross memory mode: Any PASN, any HASN, any SASN
AMODE: 31-bit
ASC mode: Primary or access register (AR)
Interrupt status: Enabled for I/O and external interrupts
DSPSERV macro for hiperspaces

Locks:
No locks held

Control parameters:
Must be in the primary address space.

Programming requirements
If your program is in AR mode, issue the SYSSTATE ASCENV=AR macro before you issue DSPSERV. SYSSTATE ASCENV=AR tells the system to generate code appropriate for AR mode.

If you use the RELEASE parameter to specify a range of storage using INLIST=YES, you must use RANGLIST to specify a range list that is mapped by the IARDRL macro. For information on the IARDRL macro, see z/OS MVS Data Areas in z/OS Internet Library at http://www.ibm.com/systems/z/os/zos/bkserv/.

Restrictions
None.

Input register information
Before issuing the DSPSERV 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Some callers depend on register contents remaining the same before and after issuing a service. If the system 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 DSPSERV macro is written as follows:

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

506   z/OS V1R13.0 MVS Assembler Services Reference ABE-HSP
DSPSERV macro for hiperspaces

One or more blanks must precede DSPSERV.

DSPSERV

One or more blanks must follow DSPSERV.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STOKEN</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>TYPE=HIPERSPACE</td>
<td></td>
</tr>
<tr>
<td>NAME</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>GENNAME</td>
<td>Default: GENNAME=NO</td>
</tr>
<tr>
<td>GENNAME=NO</td>
<td></td>
</tr>
<tr>
<td>GENNAME=COND</td>
<td></td>
</tr>
<tr>
<td>GENNAME=YES</td>
<td></td>
</tr>
<tr>
<td>OUTNAME</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>START</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>BLOCKS</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>NUMBLKS</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>INLIST</td>
<td>RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>RANGLIST</td>
<td>RS-type address or register (2) - (12). Required with INLIST=YES.</td>
</tr>
<tr>
<td>NUMRANGE</td>
<td>RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>VAR</td>
<td>Default: VAR=NO</td>
</tr>
<tr>
<td>LISTVER</td>
<td>Default: IMPLIED_VERSION</td>
</tr>
<tr>
<td>LISTVER=IMPLIED_VERSION</td>
<td></td>
</tr>
<tr>
<td>LISTVER=MAX</td>
<td></td>
</tr>
<tr>
<td>LISTVER=plistver</td>
<td></td>
</tr>
</tbody>
</table>

Chapter 91. DSPSERV — Create, delete, and control hiperspaces  507
DSPSERV macro for hiperspaces

Parameters

The CREATE, RELEASE, DELETE, and EXTEND parameters, which designate the services of the DSPSERV macro, are mutually exclusive. You can select only one.

The parameters are explained as follows:

CREATE
Requests that the system create a nonshared standard hiperspace. Creating a hiperspace is somewhat like issuing a GETMAIN for storage. The entire hiperspace is in the same storage key. When you specify CREATE, you must also specify NAME, STOKEN, and TYPE=HIPERSPACE.

Optional parameters when you create a hiperspace are: OUTNAME, GENNAME, BLOCKS, ORIGIN, and NUMBLKS.

RELEASE
Requests that the system resources used to contain the user's data be returned to the system. Although the data contained in the virtual storage is discarded, the user's virtual storage itself remains and is available for further use. When you specify RELEASE, you must also specify STOKEN to identify the hiperspace, and the START and BLOCKS parameters to identify the beginning and the length of the area to be returned to the system.

The caller must own the hiperspace, and the caller's PSW key must be zero or equal to the key of the storage the system is to release. Otherwise, the system abends the caller.

Pages that are released through DSPSERV RELEASE do not occupy space in central, expanded, or auxiliary storage. These pages are available for further use and contain hexadecimal zeros.

DELETE
Requests that the system delete a hiperspace. STOKEN is the only required parameter on the DELETE request.

A problem state or PSW key 8 through F caller must own the hiperspace, and its PSW key must be zero or equal to the storage key of the hiperspace the system is to release.

EXTEND
Requests that the system increase the current size of a hiperspace. Use EXTEND only for a hiperspace that was created with an initial size smaller than a maximum size. Before a caller can reference storage beyond the current size, the caller must use EXTEND to increase the storage that is available. If a caller references hiperspace storage beyond the current size, the system rejects the request; it terminates the caller with an 0C4 abend code.

STOKEN (identifying the hiperspace) and BLOCKS (specifying the size of the increase) are required on the EXTEND request. VAR (requesting a variable extension) and NUMBLKS (requesting the size of the extension) are optional parameters.

For the problem state and PSW key 8 through F caller, the TCB that represents it must own the hiperspace.
DSPSERV macro for hiperspaces

The system rejects the EXTEND request if you specified VAR=NO (or took the default) and the extended size would:

- Exceed the maximum size specified when the hiperspace was created.
- For a hiperspace with a storage key greater than 7, extend the cumulative data space and hiperspace totals beyond the installation limits for the owning address space.

\STOKEN=\stoken-addr
Specifies the address of the eight-byte STOKEN for the hiperspace being created, deleted, or released.

DSPSERV CREATE returns the STOKEN; STOKEN is required input for all other requests.

\TYPE=HIPERSPACE
Specifies that the system is to create a standard hiperspace rather than a data space.

\NAME=\name-addr
Specifies the address of the eight-byte variable or constant that contains the name of the hiperspace. NAME is required for DSPSERV CREATE.

Hiperspace™ names are from one to eight bytes long. They can contain letters, numbers, and @, #, and $, but they cannot contain embedded blanks. Names that contain fewer than eight bytes must be left-justified and padded on the right with blanks.

Hiperspace and data space names must be unique within the home address space of the owner. No other hiperspace or data space in the home address space can have the same name. Therefore, in choosing names for your hiperspaces, you must avoid using the same names that IBM might use for hiperspaces. Do not use the following names:

- Names that begin with A through I.
- Names that begin with a numeral or with SYS.

How to choose names for your hiperspaces:

Use hiperspace names that begin with @, #, $, or the letters J through Z, with the exception of SYS. The system abends problem state programs that begin names with SYS.

To ensure that the names for your hiperspaces are unique, ask the system to generate a unique name. See the GENNAME parameter.

\GENNAME=\genname
\GENNAME=NO
\GENNAME=COND
\GENNAME=YES
Specifies whether or not you want the system to generate a name for the hiperspace to ensure that all names are unique within the address space. The system generates a name by adding a 5-character prefix (consisting of a numeral followed by four characters) to the first three characters of the name you supply on the NAME parameter. For example, if you supply ‘XYZDATA’ on the NAME parameter, the name becomes ‘nCCCCXYZ’ where ‘n’ is the numeral, ‘CCCC’ is the 4-character string generated by the system, and XYZ comes from the name you supplied on NAME. See NAME for more information about naming conventions.

GENNAME=NO The system does not generate a name. You must supply a name unique within the address space. GENNAME=NO is the default.
DSPSERV macro for hiperspaces

GENNAME=COND The system generates a unique name only if you supply a name that is already being used. Otherwise, the system uses the name you supply.

GENNAME=YES The system takes the name you supply on the NAME parameter and makes it unique.

If you want the system to return the unique name it generates, use the OUTNAME parameter.

OUTNAME=outname-addr
Specifies the address of the eight-byte variable where the system returns the name it generated for the hiperspace. The generated name of the hiperspace if you specify GENNAME=YES or GENNAME=COND. The OUTNAME parameter is optional on DSPSERV CREATE.

START=start-addr
Specifies the address of a four-byte variable containing the beginning address of a block of storage in a hiperspace. The address must be on a four-kilobyte boundary. A block is a unit of 4K bytes. START is required on a RELEASE request.

BLOCKS=(max-addr, init-addr)
BLOCKS=(max, init)
BLOCKS=max
BLOCKS=(0, init)
BLOCKS=0
BLOCKS=(0, init-addr)
BLOCKS=size-addr
BLOCKS=size

Specifies the size of a hiperspace or the size of an area within a hiperspace. BLOCKS is required for all requests except for DSPSERV DELETE.

For a CREATE request, specifies the maximum size (in blocks) to which the hiperspace can expand (max-addr or max) and the initial size of the hiperspace (init-addr or init). A block is a unit of 4K bytes. You cannot extend the hiperspace beyond its maximum size.

max-addr specifies the address of a field that contains the maximum size of the hiperspace to be created. max is the number of blocks (up to 524,288) to be used for the hiperspace.

init-addr specifies the address of the initial size of the hiperspace. init is the number of blocks to be used as the initial size. If the initial size you specify exceeds or equals the maximum size, then the initial size becomes the maximum size.

0 specifies the default size, either the installation default or the IBM-defined default. The IBM-defined default maximum is 239 blocks. Your installation can use the installation exit IEFUSI to change the IBM default. The system returns the maximum size at the location identified by NUMBLKS.

If you do not code the BLOCKS parameter on the CREATE request, the system uses BLOCKS=0, setting the initial size and the maximum size equal to the installation (or IBM) default.

For a RELEASE request, BLOCKS and START are required parameters that define contiguous storage (in 4K blocks) that the system is to release. BLOCKS specifies the size of an area to be released (size-addr or size). The minimum size is 1 block and the maximum is 524,288 blocks (2 gigabytes).
For an EXTEND request, BLOCKS is a required parameter that defines the amount of increase of the current size of the hiperspace.

\texttt{,ORIGIN=origin-addr}

- Specifies the address of the four-byte variable that contains the lowest address (either zero or 4096) of the new hiperspace. The system returns the beginning address of the hiperspace at \texttt{origin-addr}. The system tries to start all hiperspaces at origin zero; on some processors, however, the origin is 4096. ORIGIN is an optional parameter for DSPSERV CREATE.

\texttt{,NUMBLKS=numblks-addr}

- Specifies the address of the four-byte area where the system returns one of the following:
  - For DSPSERV CREATE, the maximum size (in blocks) of the newly created hiperspace.
  - For DSPSERV EXTEND, the size by which the system extended the hiperspace.

The NUMBLKS parameter is an optional parameter on DSPSERV CREATE and DSPSERV EXTEND.

If, when you create a hiperspace, you specify BLOCKS=0 or do not specify the BLOCKS parameter, the system uses the default that your installation established in the installation exit IEFUSI. The system returns this default value at \texttt{numblks-addr}.

\texttt{,INLIST=NO}
\texttt{,INLIST=YES}

- Specifies whether a range is included (YES). The default is \texttt{INLIST=NO}. If you specify YES, you must also specify the RANGLIST parameter.

\texttt{,RANGLIST=rangelist-addr}

- Specifies the name (RS-type) or address (in register 2-12) of a required input fullword that contains the address of the range list. The range list consists of a number of entries (specified by NUMRANGE); each entry is 8 bytes long. A mapping of each entry is provided through the mapping macro IARDRL.

\texttt{,NUMRANGE=numrange_addr}

- Specifies the name (RS-type) or address (in register 2-12) of an optional parameter that provides the number of entries in the supplied RANGLIST. The maximum value may not exceed 16. The default is 1.

\texttt{,VAR=NO}
\texttt{,VAR=YES}

- Specifies whether your request for the system to extend the amount of storage available in a hiperspace is a variable request. When you use DSPSERV EXTEND for a hiperspace, the system might not be able to extend the hiperspace by the amount you request, because that amount might cause the system to exceed one of the following:
  - The maximum size of the hiperspace, as specified on the BLOCKS parameter when the hiperspace was created.
  - For a hiperspace with storage key 8-F, the limit of combined data space and hiperspace storage with storage key 8-F for an address space. (The installation established this limit on the IEFUSI installation exit, or took the IBM default.)

If you specify VAR=YES (the variable request) and the system cannot satisfy your request, the system extends the hiperspace to one of the following sizes, depending on which is smaller:
DSPSERV macro for hiperspaces

- The maximum size specified on the BLOCKS parameter when the hiperspace was created.
- The largest size that would still keep the combined total of data space and hiperspace storage within the limits established by the installation for an address space.

If you specify VAR=NO (the default), the system:
- Abends the caller if the extended size would exceed the maximum size specified when the hiperspace was created.
- Rejects the request if the hiperspace has storage key 8 - F and the request would extend the cumulative data space and hiperspace totals beyond the installation limits for an address space.

If you use the NUMBLKS parameter, the system returns the size by which the system extends the hiperspace.

\textbf{PLISTVER=IMPLIED\_VERSION, PLISTVER=MAX, PLISTVER=plistver}

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

- \textbf{IMPLIED\_VERSION}, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED\_VERSION is the default.
- \textbf{MAX}, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.
  
  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
- \textbf{0}, if you use the currently available parameters.

To code, specify in this input parameter one of the following:
- \textbf{IMPLIED\_VERSION}
- \textbf{MAX}
- A decimal value of 0

\textbf{MFS=S}

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

**ABEND codes**

DSPSERV might abnormally terminate with abend code X'01D'. See \texttt{z/OS MVS System Codes} for an explanation and programmer response.
## DSPSERV macro for hiperspaces

### Return and reason codes

Hexadecimal return and reason codes from DSPSERV CREATE:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00          | —           | **Meaning:** DSPSERV CREATE completed successfully.  
**Action:** None. |
| 04          | xx000Cxx    | **Meaning:** Program error. DSPSERV CREATE completed successfully. You specified a size of 2 gigabytes (524,288 blocks). However, because the processor did not support a hiperspace with zero origin, a hiperspace of one less block (524,287 blocks) was created.  
**Action:** None required. However, you should verify that your program correctly accounts for the nonzero origin of the hiperspace. |
| 08          | xx0005xx    | **Meaning:** Program error. Creation of the hiperspace would violate installation criteria. See the IEFUSI installation exit in [z/OS MVS Installation Exits](https://www.ibm.com/support/knowledgecenter/STXKQY_2.2.0/com.ibm.mvs.doc/тензr/zeosinst.html).  
**Action:** Check with your system programmer for local restrictions on the creation and use of hiperspaces. |
| 08          | xx0009xx    | **Meaning:** Program error. The specified hiperspace name is not unique within the address space.  
**Action:** Check that the hiperspace name is not already in use by another active hiperspace. Change the hiperspace name or specify the GENNAME parameter on the DSPSERV macro to get the system to generate a unique name. |
| 08          | xx0012xx    | **Meaning:** Environmental error. The system's set of generated names for data spaces and hiperspaces has been temporarily depleted.  
**Action:** Retry the job one or more times during a period of lower system usage. If the problem persists, consult your system programmer, who might be able to tune the system so that more names are available for use. |
| 0C          | xx0006xx    | **Meaning:** Environmental error. The system cannot create any additional data spaces at this time because of a shortage of resources. For reason code 6C000611, an ASTE could not be obtained for the requested data space. If the request is for a SCOPE=COMMON data space, this may mean there are already as many SCOPE=COMMON data spaces in the system as are allowed by the MAXCAD parameter.  
**Action:** Retry the job one or more times during a period of lower system usage. If the problem persists, consult your system programmer, who might be able to tune the system so that resources will not become depleted. |
DSPSERV macro for hiperspaces

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C          | xx0007xx    | **Meaning**: System error. The system cannot obtain addressability to its own hiperspaces.  
**Action**: Record the return and reason code and supply it to the appropriate IBM support personnel. |

Hexadecimal return and reason codes from DSPSERV EXTEND:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00          | —           | **Meaning**: DSPSERV EXTEND completed successfully.  
**Action**: None. |
| 08          | xx0502xx    | **Meaning**: Environmental error. Extending the hiperspace size would cause the data space and hiperspace limits for the address space to be exceeded.  
**Action**: Check with your system programmer, who might be able to tune the system so that the function is made available to your program. |
| 08          | xx0503xx    | **Meaning**: Program error. You are using VAR=YES to extend the current size of the hiperspace; however, the hiperspace is already the maximum size.  
**Action**: None required. However, if your program requires more storage, you should consider creating an additional hiperspace. |

The caller of DSPSERV does not receive any return codes for the RELEASE and DELETE services.

**Example**

Create a hiperspace named TEMP with a size of 10 million bytes.

```assembly
DSPSERV CREATE,NAME=HSPCNAME,STOKEN=HSPCSTKN,X TYPE=HIPERSPACE,BLOCKS=HSPBLCKS,ORIGIN=HSPCORG
```

```assembly
HSPCNAME DC CL8'TEMP' HIPERSPACE NAME
HSPCSTKN DS CL8 HIPERSPACE STOKEN
HSPCORG DS F HIPERSPACE ORIGIN RETURNED
HSPCSIZE EQU 10000000 NUMBER OF BLOCKS NEEDED FOR A 10 MILLION BYTE HIPERSPACE
HSPBLCKS DC A((HSPCSIZE+4095)/4096) NUMBER OF BLOCKS NEEDED FOR A 10 MILLION BYTE HIPERSPACE
```

**DSPSERV—List form**

Use the list form of the DSPSERV macro to construct a nonexecutable control program parameter list.

**Syntax**

The list form of the DSPSERV macro is written as follows:
DSPSERV macro for hiperspaces

name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede DSPSERV.

DSPSERV

One or more blanks must follow DSPSERV.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=plistver

Default: IMPLIED_VERSION
plistver: 0

,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 as follows:

,MF=(L,list addr)
,MF=(L,list addr,attr)

Specify the list form of the DSPSERV macro. list addr defines the area that the system is to use for the parameter list.

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

DSPSERV—Execute form

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

Syntax

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

name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede DSPSERV.

DSPSERV

One or more blanks must follow DSPSERV.
DSPSERV macro for hiperspaces

Valid parameters (Required parameters are underlined.)

CREATE
STOKEN, NAME, TYPE, GENNAME, OUTNAME,
BLOCKS, ORIGIN, and NUMBLKS

RELEASE
STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE

DELETE
STOKEN

EXTEND
STOKEN, BLOCKS, VAR, NUMBLKS

STOKEN=stoken-addr

TYPE=HIPERSPACE

NAME=name-addr

GENNAME=NO

GENNAME=COND

GENNAME=YES

OUTNAME=outname-addr

START=start-addr

BLOCKS=(max-addr,init-addr)

BLOCKS=(max,init)

BLOCKS=max

BLOCKS=(0,init)

BLOCKS=0

BLOCKS=(0,init-addr)

BLOCKS=(size-addr)

BLOCKS=size

ORIGIN=origin-addr

NUMBLKS=numblks-addr

INLIST=NO

INLIST=YES

RANGLIST=rangelist-addr

NUMRANGE=numrange-addr

VAR=NO

VAR=YES

PLISTVER=IMPLIED_VERSION

PLISTVER=MAX

PLISTVER=plistver

MF=(E,list addr)

MF=(E,list addr,COMPLETE)
Parameters

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

\[ \text{MF} = (E, \text{list addr}) \]
\[ \text{MF} = (E, \text{list addr}, \text{COMPLETE}) \]

Specifies the execute form of the DSPSERV macro. \text{list addr} defines the area that the system uses for the parameter list.

\text{COMPLETE} specifies that the system is to check for required parameters and supply optional parameters that are not specified.
DSPSERV macro for hiperspaces
Chapter 92. EDTINFO — Obtain eligible device table information

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

The EDTINFO macro performs the following functions:
- Check groups (CHKGRPS)
- Check units (CHKUNIT)
- Return unit name (RTNUNIT)
- Return unit control block (UCB) addresses for static and installation-static devices defined below 16 megabytes with 3-digit device numbers (RTNUCBA)
- Return group ID (RTNGRID)
- Return attributes (RTNATTR)
- Return unit names for a device class (RTNNAMD)
- Return UCB device number list (RTNDEVN)
- Return maximum eligible device type (MAXELIG)
- Return default unit-affinity-ignored unit name (RTNUNAFF)

Any one of these functions, or any combination of them, may be specified on each invocation of the EDTINFO macro.

Notes:
1. If you specify both RTNUNIT and MAXELIG, the variable specified by OUTUNIT will contain the results of the MAXELIG function.
2. If you specify both RTNUNIT and RTNUNAFF, the variable specified by OUTUNIT will contain the results of the RTNUNIT function.
3. If you specify both MAXELIG and RTNUNAFF, the variable specified by OUTUNIT or OUTDEV will contain the results of the MAXELIG function.

Environment
The requirements for the caller are:
- Minimum authorization: Problem state and any PSW key.
- Dispatchable unit mode: Task
- Cross memory mode: PASN=HASN=SASN or PASN=HASN=SASN
- AMODE: 24- or 31- bit
- ASC mode: Primary or access register (AR)
- Interrupt status: Enabled for I/O and external interrupts
- Locks: No locks held
- Control parameters: Must be in the primary address space. This includes data areas whose address is passed to EDTINFO.

Programming requirements
None.

Restrictions
None.
Input register information

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

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR 15 contains a return code of 04 or 08; otherwise, used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Performance implications

None.

Syntax

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

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

/bslash
One or more blanks must precede EDTINFO.

EDTINFO
/bslash
One or more blanks must follow EDTINFO.

CHKGPRS
Note: At least one of these functions is required: CHKGPRS, CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR, RTNNAMD, RTNDEVN, MAXELIG, RTNUNAFF. If more than one of these is specified, a comma must be coded between each of the keywords.

CHKUNIT

RTNUNIT

RTNUCBA
Note: See the tables following this diagram for information on parameter usage with these functions.
```
Chapter 92. EDTINFO — Obtain eligible device table information

EDTINFO macro

RTNGRID
RTNATTR
RTNNAMD
RTNDEVN
MAXELIG
RTNUNAFF

.DEVCOUNT=devcount addr  devcount addr: RS-type address or register (2) - (12).
.DEVLIST=devlist addr  devlist addr: RS-type address or register (2) - (12).
.DEVSTAT=devstat addr  devstat addr: RS-type address or register (2) - (12).
.UNITNAME=unitname addr  unitname addr: RS-type address or register (2) - (12).
.DEVTYPE=devtype addr  devtype addr: RS-type address or register (2) - (12).
.SUBPOOL=subpool addr  subpool addr: RS-type address or register (2) - (12).
.UBALIST=ucbalist addr  ucbalist addr: RS-type address or register (2) - (12).
.URLIST=ucblist addr  ucblist addr: RS-type address or register (2) - (12).
.GRIDLIST=gridlist addr  gridlist addr: RS-type address or register (2) - (12).
.ATTRAREA=attrarea addr  attrarea addr: RX-type address or register (2) - (12).
.DEVCLASS=devclass addr  devclass addr: RS-type address or register (2) - (12).
.NAMELIST=namelist addr  namelist addr: RS-type address or register (2) - (12).

.DYNAMIC=YES
.DYNAMIC=NO  Default: DYNAMIC=NO

.LOC=BELOW
.LOC=ANY  Default: LOC=BELOW

.RANGE=ALL
.RANGE=3DIGIT  Default: RANGE=3DIGIT

.DEVNLIST=devnlist addr  devnlist addr: RS-type address or register (2) - (12).
.RECMODE=recmode addr  recmode addr: RS-type address or register (2) - (12).
.DENSITY=density addr  density addr: RS-type address or register (2) - (12).
.OUTUNIT=outunit addr  outunit addr: RS-type address or register (2) - (12).
.OUTDEV=outdev addr  outdev addr: RS-type address or register (2) - (12).
.IOCTOKEN=ioctoken addr  ioctoken addr: RX-type address or register (2) - (12).
.RETCODE=retcode addr  retcode addr: RX-type address or register (2) - (12).
The following tables show how the parameters may be specified with the
CHKGRRPS, CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR, RTNAMD,
RTNDEVN, MAXELIG, and RTNUNAFF functions.

The IOCTOKEN, RETCODE, and RSNCODE parameters are optional with any of
the functions.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>CHKGRRPS</th>
<th>CHKUNIT</th>
<th>RTNUNIT</th>
<th>RTNUCBA</th>
<th>RTNGRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVCOUNT</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVLIST</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVSTAT</td>
<td>optional</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UNITNAME</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVTYPY</td>
<td>not valid</td>
<td>DEVTYPE or UNITNAME required</td>
<td>required</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
</tr>
<tr>
<td>SUBPOOL</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBALIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
</tr>
<tr>
<td>GRIDLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
</tr>
<tr>
<td>ATTRAREA</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVCLASS</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>NAMELIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>LOC</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RANGE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVNLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RECMODE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DENSITY</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTUNIT</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTDEV</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RTNATTR</th>
<th>RTNAMD</th>
<th>RTNDEVN</th>
<th>MAXELIG</th>
<th>RTNUNAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVCOUNT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVSTAT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UNITNAME</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
</tr>
</tbody>
</table>
### EDTINFO macro

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RTNATTR</th>
<th>RTNNAME</th>
<th>RTNDVN</th>
<th>MAXELIG</th>
<th>RTNUAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVTYPE</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
<td>DEVTYPE or UNITNAME required</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
</tr>
<tr>
<td>SUBPOOL</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBALIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>GRIDLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>ATTRAREA</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVCLASS</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>NAMELIST</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>LOC</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RANGE</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVLNLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RECMODE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>DENSITY</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTUNIT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>OUTUNIT or OUTDEV required</td>
<td>OUTUNIT or OUTDEV required</td>
</tr>
<tr>
<td>OUTDEV</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>OUTDEV or OUTUNIT required</td>
<td>OUTDEV or OUTUNIT required</td>
</tr>
</tbody>
</table>

**Note:** Code the parameters as indicated for each of the function keywords when you specify multiple functions. For example, assume that you specify the CHKGRPS and RTNATTR functions. The CHKGRPS function requires DEVCOUNT and DEVLIST to be specified, and the RTNATTR function requires UNITNAME or DEVTYPE to be specified. Because DEVCOUNT and DEVLIST are required with CHKGRPS, you must code them if you specify CHKGRPS, even though DEVCOUNT and DEVLIST are not valid with RTNATTR. Similarly, UNITNAME or DEVTYPE is required with RTNATTR and must be coded, even though neither one is valid with CHKGRPS.

### Parameters

The parameters are explained as follows:

**CHKGRPS**

Specifies that the EDTINFO service should determine whether the specified device numbers constitute a valid allocation group. The device numbers are specified by the DEVCOUNT, DEVLIST, and, optionally, DEVSTAT parameters, and are a valid allocation group if either of the following is true:

- For any allocation group in the EDT that contains at least one of the device numbers specified in the input device number list, **all** of the device numbers in that group in the EDT are contained in the input device number list.
- None of the allocation groups in the EDT contain any of the device numbers specified in the input device number list.
If neither of these is the case, the device numbers are not a valid allocation group.

**Note:** In addition to generating a return code and reason code, EDTINFO sets bit 0 in the flag byte of any entry in the device number list or the device status list, if present, if the entry corresponds to a device number that is not valid.

**CHKUNIT**

Specifies that the EDTINFO service should determine whether the input device numbers correspond to the specified unit name. The input device numbers are specified by the UNITNAME or DEVTYPE, DEVCOUNT, DEVLIST, and, optionally, DEVSTAT parameters. The unit name is the EBCDIC representation of the IBM generic device type (for example, 3380) or the esoteric group name (for example, TAPE) from the EDT.

**Notes:**

1. In addition to generating a return code and reason code, EDTINFO sets bit 0 in the flag byte of any entry in the device number list or the device status list, if present, if the entry corresponds to a device number that is not valid.

2. If all of the device numbers are valid but not all of them match the unit name or the device type specified as input, EDTINFO in addition to generating a return and reason code, sets bit 1 in the flag byte of any entry in the device number list or the device status list, if present, if the entry does not correspond to the input unit name or device type.

**RTNUNIT**

Specifies that the EDTINFO service should return the unit name associated with the UCB device type that is provided as input in the DEVTYPE parameter. The unit name is returned in the storage specified by the OUTUNIT parameter.

**Note:** Do not use the RTNUNIT parameter to determine whether a returned unit name is a generic CTC device or an esoteric group name that contains CTC devices. Instead, use the RTNATTR parameter for this purpose.

**RTNUCBA**

Specifies that the EDTINFO service should return a list of pointers to UCBs associated with the unit name or device type provided as input in the UNITNAME or DEVTYPE parameter. EDTINFO returns UCB addresses only for static and installation-static below 16 megabyte UCBs with 3-digit device numbers. The address of the UCB pointer list is returned in the storage specified by the UCBALIST parameter. You can specify the subpool in which to obtain storage by using the SUBPOOL list.

**Note:** You can use the RTNDEVN parameter instead to obtain a list of device numbers belonging to a specified unit name or UCB device type, including dynamic devices, 4-digit devices and devices described by UCBs residing above the 16-megabyte line. Then the UCBINFO macro can be used to obtain selected UCB device information for a given device number.

If your program is authorized, running in supervisor state or with a program key mask of 0-7, you can use the UCBLOOK macro to obtain the actual UCB address from a given device number. See [z/OS](#)
EDTINFO macro

Specifies that the EDTINFO service should return the allocation group ID corresponding to each UCB address specified by the UCBLIST parameter. The address of the group ID list is returned in the storage specified by the GRIDLIST parameter.

Specifies that the EDTINFO service should return general information about the unit name or device type specified in the UNITNAME or DEVTYPE parameter. The information is returned in the storage specified by the ATTRAREA parameter.

Specifies that the EDTINFO service should return a list of IBM generic device types (for example, 3380) or esoteric group names (for example, TAPE) associated with the input device class specified in the DEVCLASS parameter. The address of the unit name list is returned in the storage specified by the NAMELIST parameter. You can specify the subpool in which to obtain storage by using the SUBPOOL parameter.

Specifies that the EDTINFO service should return the UCB device number list associated with the unit name or UCB device type specified by the UNITNAME or DEVTYPE parameter. The address of the device number list is returned at the address specified by the DEVNLIST parameter. By using the DYNAMIC parameter, you can specify that devices defined to the system as dynamic are to be included in the list. By using the RANGE parameter, you can include 4-digit device numbers in the returned UCB device number list. By using the LOC parameter, you can include devices with actual above 16 megabyte UCBs in the returned UCB device number list.

Specifies that the EDTINFO service should determine the maximum eligible device type (for the allocation and cataloging of a data set on a tape device) associated with the unit name or device type, recording mode, and density provided as input. The maximum eligible device type is the tape device type that contains the greatest number of eligible devices compatible with the specified recording mode and density. You specify the unit name or device type in the UNITNAME or DEVTYPE parameters. The recording mode and density are specified in the RECMODE and DENSITY parameters. EDTINFO returns the maximum eligible device type in the OUTUNIT or OUTDEV parameter, depending on which one you specify.

Specifies that the EDTINFO service should return the default unit-affinity-ignored unit name that was provided on the UNITAFF subparameter of the UNIT parameter in the ALLOCxx parmlib member, or defaulted by the system. The unit name is returned in the storage specified by the OUTUNIT parameter, or the device type is returned in the storage specified by the OUTDEV parameter, depending on which one you specify.

Specifies the fullword input field that contains the number of entries in the input device number list and the optional output device status list.
EDTINFO macro

,DEVLIST=devlist addr
Specifies the address of an input pointer that contains the address of the device number list. This list can be in two different formats:

- The first format is used for 3-digit device numbers. The format consists of an array of 4-byte entries. The first 3 bytes contain the EBCDIC device number and the last byte is a flag byte containing output flags. Bit 0 in the flag byte indicates the validity of the device number: If the bit is set to 1, the device number is not valid. Bit 1 in the flag byte indicates whether the device number is associated with the unit name or the device type specified as input: If the bit is set to 1, the device number is not associated with the unit name or device type.
- The second format is used for 4-digit device numbers; DEVSTAT must also be specified. Each entry in the format contains a 4-byte EBCDIC device number. The status byte is in the device status array provided by the DEVSTAT parameter.

,DEVSTAT=devstat addr
Specifies the address of an input pointer that contains the address of the output device status list. This optional list consists of an array of 2-byte entries that are parallel to the input device number list. In each entry, the first byte contains output flags and the second byte is reserved for IBM use. Bit 0 in the flag byte indicates the validity of the device number contained in the device number list. If the bit is set to 1, the device number is not valid. Bit 1 in the flag byte indicates whether the device number contained in the device number list is associated with the unit name or the device type specified as input. If that bit is set to 1, the device number is not associated with the input unit name or device type.

,UNITNAME=unitname addr
,DEVTYPE=devtype addr
Specifies either the 8-character input field that contains the unit name (UNITNAME=unitname addr) or specifies the 4-character input field that contains the 4-byte UCB device type (DEVTYPE=devtype addr).

,SUBPOOL=subpool addr
Specifies a 1-byte input field that indicates in which subpool the storage should be obtained. If you do not specify SUBPOOL, the default is subpool 230 if the caller is authorized, and subpool 0 if the caller is not authorized. The caller is responsible for freeing the storage once it is no longer required.

,UCBALIST=ucbalist addr
Specifies the address of an output pointer that is to contain the address of the UCB pointer list. The pointer list format is as follows:

- an 8 byte header containing
  - a 1-byte field indicating the subpool in which the storage resides
  - a 3-byte field containing the size of the pointer list (including the header)
  - a 4-byte field containing the number of entries in the list.
- an array of 4-byte entries containing the actual UCB addresses (for below 16 megabyte static and installation-static UCBs with 3-digit device numbers only).

,UCBLIST=ucblist addr
Specifies the address of an input pointer that contains the address of the UCB pointer list. This list consists of a 4-byte header containing the number of entries in the list followed by an array of 4-byte entries containing the actual or captured UCB addresses.
EDTINFO macro

\`GRIDLIST\'=gridlist addr
Specifies the address of an input pointer that contains the address of the group ID list. This list is an array of 4-byte entries that parallel the input UCB pointer list entries and contain the group ID associated with each UCB.

\`ATTRAREA\'=attrarea addr
Specifies the address of a 10-character output field in which general information about the unit name or device type (specified by the UNITNAME or DEVTYPE parameter) is returned. The contents of ATTRAREA are:

<table>
<thead>
<tr>
<th>Byte</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Length of the attribute area (X'0A'). You must fill in this byte prior to issuing EDTINFO.</td>
</tr>
<tr>
<td>1-2</td>
<td>Flags describing the unit name:</td>
</tr>
<tr>
<td>Bit</td>
<td>Meaning</td>
</tr>
<tr>
<td>0</td>
<td>If bit is on, the unit name is an esoteric group name.</td>
</tr>
<tr>
<td>1</td>
<td>If bit is on, the unit name is VIO-eligible.</td>
</tr>
<tr>
<td>2</td>
<td>Not part of the programming interface.</td>
</tr>
<tr>
<td>3</td>
<td>If bit is on, the unit name contains TP class devices.</td>
</tr>
<tr>
<td>4-15</td>
<td>Not part of the programming interface.</td>
</tr>
<tr>
<td>3</td>
<td>Number of device classes in the unit name.</td>
</tr>
<tr>
<td>4-7</td>
<td>Number of generic device types in the unit name.</td>
</tr>
<tr>
<td>8-9</td>
<td>Not part of the programming interface.</td>
</tr>
</tbody>
</table>

\`DEVCLASS\'=devclass addr
Specifies the address of a 1-character input field that contains the device class in hexadecimal.

\`NAMELIST\'=namelist addr
Specifies the address of an output pointer that is to contain the address of the unit name list. The format of the unit name list is as follows:
- an 8-byte header containing
  - a 1-byte field indicating the subpool in which the storage resides
  - a 3-byte field containing the size of the unit name list (including the header)
  - a 4-byte field containing the number of entries in the list
- an array of 8-byte entries containing the actual unit names.

\`DYNAMIC\'=YES
\`DYNAMIC\'=NO
Specifies whether dynamic devices should (DYNAMIC=YES) or should not (DYNAMIC=NO) be included in the device number list. If you specify DYNAMIC=NO, only static and installation-static devices are included in the list.

\`LOC\'=ANY
\`LOC\'=BELOW
Specifies whether the output device number list should be restricted to devices with below 16 megabyte UCBs (LOC=BELOW) or should also include devices with above 16 megabyte UCBs (LOC=ANY) when you specify the RTNDEVN parameter.

\`RANGE\'=ALL
EDTINFO macro

\texttt{,RANGE=3DIGIT}

Specifies whether all devices (RANGE=ALL) or only those devices with device numbers of 3 digits or less (RANGE=3DIGIT) should be included in the output device number list.

\texttt{,DEVNLIST=devnlist addr}

Specifies the address of an output pointer that is to contain the address of the output device number list. In other words, DEVNLIST is a pointer to a pointer to the output device number list. The format of the device number list is as follows:

- a 4-byte field containing the size of the list (including the header)
- a 4-byte field containing the number of entries in the list
- an array of 4-byte entries containing the actual EBCDIC device numbers.

This storage must be obtained by the caller prior to invoking the EDTINFO macro and must reside in the caller's key. The caller must store the length of the list into the header before invoking the macro. If there is not enough storage to contain all of the entries, the following occurs:

- a return code of 8 and a reason code of 4 are returned
- the number of entries is filled in
- no EBCDIC device numbers are returned.

\texttt{,IOCTOKEN=ioctoken addr}

Specifies a 48-character area for the MVS I/O configuration token. If the current EDT definition is not consistent with the token specified as input by \texttt{ioctoken addr}, the caller is notified through a return code.

If the input specified by \texttt{ioctoken addr} is set to binary zeros, EDTINFO sets IOCTOKEN to the current MVS I/O configuration token.

\texttt{,RECMODE=recmode addr}

Specifies the address of an 8-bit input that indicates the recording mode.

\texttt{,DENSITY=density addr}

Specifies the address of an 8-bit input that indicates the density.

\texttt{,OUTUNIT=outunit addr}
\texttt{,OUTDEV=outdev addr}

Specifies the address of an 8-character field where EDTINFO returns the unit name (OUTUNIT=\texttt{outunit addr}) or specifies the address of a 4-character field where EDTINFO returns the 4-byte device type (OUTDEV=\texttt{outdev addr}).

\texttt{,RETCODE=retcode addr}

Specifies the fullword location where the system is to store the return code. The return code is also in GPR 15.

\texttt{,RSNCODE=rsncode addr}

Specifies the fullword location where the system is to store the reason code. The reason code is also in GPR 0.

Return and reason codes

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

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>The requested function or functions were performed and no reason code information has been returned.</td>
</tr>
</tbody>
</table>
**EDTINFO macro**

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>04</td>
<td>The requested function or functions were performed and information has been returned, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in rsncode addr, if you coded RSNCODE).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>The input device numbers do not belong to the same group.</td>
</tr>
<tr>
<td>02</td>
<td>One or more of the input device numbers does not belong to the input unit name or device type.</td>
</tr>
<tr>
<td>03</td>
<td>The input unit name was valid but no units matching the specified or defaulted selection criteria were found. No UCB addresses or device numbers have been returned.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>08</td>
<td>There is data in the input parameter list that is not valid, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in rsncode addr, if you coded RSNCODE).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reason Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>The input unit name could not be found in the EDT.</td>
</tr>
<tr>
<td>02</td>
<td>The input device type could not be found in the EDT.</td>
</tr>
<tr>
<td>03</td>
<td>One or more of the input device numbers is invalid.</td>
</tr>
<tr>
<td>04</td>
<td>The caller did not provide sufficient storage for the returned information.</td>
</tr>
<tr>
<td>05</td>
<td>The MAXELIG function requires a generic device type as input, but the input specified does not represent a generic device type.</td>
</tr>
<tr>
<td>06</td>
<td>The caller did not request any functions.</td>
</tr>
<tr>
<td>07</td>
<td>The caller requested functions that are not valid.</td>
</tr>
<tr>
<td>08</td>
<td>For a required input, the caller specified a value that is not valid. For example, other functions were specified with a function that requires no other function requests.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>A configuration change has occurred and the input I/O configuration token does not match the current token.</td>
</tr>
<tr>
<td>10</td>
<td>Storage could not be obtained for the request.</td>
</tr>
<tr>
<td>18</td>
<td>An unexpected system error occurred.</td>
</tr>
</tbody>
</table>

**Note:** When you specify multiple functions, the system returns the return code with the highest numerical value, and its associated reason code.

**Example 1**

Obtain the attributes for the device whose unit name is contained in UNIT_NAME. Return the information in ATTRIBUTE_AREA.

EDTINFO RTNATTR,UNITNAME=UNIT_NAME,ATTRAREA=ATTRIBUTE_AREA

**Example 2**

Obtain the list of device numbers for the device type specified in DEVICE_TYPE. Include dynamic devices in the list. Return the list in the area pointed to by DEVICE_LIST_PTR.
EDTINFO macro

EDTINFO RTNDEVN,DYNAMIC=YES,DEVTYPE=DEVICE_TYPE, X
   DEVNLIST=DEVICE_LIST_PTR

Example 3

Determine whether the list of device numbers specified by DEVICE_LIST_PTR is a
valid allocation group. DEVICE_COUNT specifies a field containing the number of
entries in the list. Use the IOCTOKEN parameter to return the current MVS I/O
configuration token in CONFIG_TOKEN. The status of the devices is returned in the
list specified by STATUS_LIST_PTR.

Following some other processing, return the allocation group ID that corresponds to
each UCB address found in the list specified by UCB_LIST_PTR. Return the list of
group IDs in the area specified by GRID_LIST_PTR. Use the IOCTOKEN
parameter, specifying the previously-obtained MVS I/O configuration token as input
in CONFIG_TOKEN, to determine whether the I/O configuration has changed since
EDTINFO was issued.

EDTINFO CHKGRPS,DEVCOUNT=DEVICE_COUNT, X
   DEVLIST=DEVICE_LIST_PTR,IOCTOKEN=CONFIG_TOKEN, X
   DEVSTAT=STATUS_LIST_PTR.

EDTINFO RTNGRID,UCBLIST=UCB_LIST_PTR, X
   GRIDLIST=GRID_LIST_PTR,IOCTOKEN=CONFIG_TOKEN

Example 4

Determine whether the list of device numbers specified by DEVICE_LIST_PTR is a
valid allocation group, and determine if these device numbers correspond to the unit
name in the EDT. DEVICE_COUNT specifies a field containing the number of
entries in the list. DEVICE_TYPE specifies a field containing the device type. Store
the return code from register 15 in RETURN_CODE, and store the reason code
from register 0 in REASON_CODE. The status of the devices is returned in the list
specified by STATUS_LIST_PTR.

EDTINFO CHKGRPS,CHKUNIT,DEVTYPE=DEVICE_TYPE, X
   DEVCOUNT=DEVICE_COUNT,DEVLIST=DEVICE_LIST_PTR, X
   RETCODE=RETURN_CODE,RSCODE=REASON_CODE, X
   DEVSTAT=STATUS_LIST_PTR

Example 5

Return (in the output device number list specified by DEVICE_LIST_PTR) the UCB
device numbers associated with the device type DEVICE_TYPE. All devices should
be included in the output device number list.

EDTINFO RTNDEVN,DEVTYPE=DEVICE_TYPE,DYNAMIC=YES X
   RANGE=ALL,LOC=ANY, X
   DEVNLIST=DEVICE_LIST_PTR

EDTINFO - List form

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

Syntax

This macro is an alternative list form macro, and requires a different technique for
using the list form as compared to the conventional list form macros. See
“Alternative list form macros” on page 12 for further information.
The list form of the EDTINFO macro is written as follows:

```
name name
```

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

EDTINFO

b

One or more blanks must precede EDTINFO.

EDTINFO

b

One or more blanks must follow EDTINFO.

```
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 as follows:

- **MF=(L,list addr)**
- **MF=(L,list addr,attr)**
- **MF=(L,list addr,0D)**

  Specifies the list form of the EDTINFO macro.

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

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

### EDTINFO - Execute form

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

### Syntax

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

```
name name
```

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

EDTINFO

b

One or more blanks must precede EDTINFO.
EDTINFO macro

One or more blanks must follow EDTINFO.

**Note:** At least one of these functions is required: CHKGRPS, CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR, RTNNAMD, RTNDEVN, MAXELIG, RTNUNAFF. If more than one of these is specified, a comma must be coded between each of the keywords.

**Note:** See the tables following this diagram for information on parameter usage with these functions.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHKGRPS</td>
<td></td>
</tr>
<tr>
<td>CHKUNIT</td>
<td></td>
</tr>
<tr>
<td>RTNUNIT</td>
<td></td>
</tr>
<tr>
<td>RTNUCBA</td>
<td></td>
</tr>
<tr>
<td>RTNGRID</td>
<td></td>
</tr>
<tr>
<td>RTNATTR</td>
<td></td>
</tr>
<tr>
<td>RTNNAMD</td>
<td></td>
</tr>
<tr>
<td>RTNDEVN</td>
<td></td>
</tr>
<tr>
<td>MAXELIG</td>
<td></td>
</tr>
<tr>
<td>RTNUNAFF</td>
<td></td>
</tr>
<tr>
<td>,DEVCOUNT=</td>
<td>devcount addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DEVLIST=</td>
<td>devlist addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DEVSTAT=</td>
<td>devstat addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,UNITNAME=</td>
<td>unitname addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DEVTYPE=</td>
<td>devtype addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,SUBPOOL=</td>
<td>subpool addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,UCBALIST=</td>
<td>ucbalist addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,UCBLIST=</td>
<td>ucblist addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,GRIDLIST=</td>
<td>gridlist addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,ATTRAREA=</td>
<td>attrarea addr: RX-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DEVCLASS=</td>
<td>devclass addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,NAMELIST=</td>
<td>namelist addr: RS-type address or register (2) - (12).</td>
</tr>
<tr>
<td>,DYNAMIC=</td>
<td>Default: DYNAMIC=NO</td>
</tr>
<tr>
<td>,LOC=BELOW</td>
<td>Default: BELOW</td>
</tr>
<tr>
<td>,LOC=ANY</td>
<td></td>
</tr>
<tr>
<td>,RANGE=ALL</td>
<td>Default: RANGE=3DIGIT</td>
</tr>
<tr>
<td>,RANGE=3DIGIT</td>
<td></td>
</tr>
</tbody>
</table>

z/OS V1R13.0 MVS Assembler Services Reference ABE-HSP
The following tables show how the parameters may be specified with the CHKGRPS, CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR, RTNNAMD, RTNDEVN, MAXELIG, or RTNUNAFF functions.

The IOCTOKEN, RETCODE, RSNCODE, and MF parameters are optional with any of the functions.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>CHKGRPS</th>
<th>CHKUNIT</th>
<th>RTNUNIT</th>
<th>RTNUCBA</th>
<th>RTNGRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVCOUNT</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVLIST</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVSTAT</td>
<td>optional</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UNITNAME</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVTYPE</td>
<td>not valid</td>
<td>DEVTYPE or UNITNAME required</td>
<td>required</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
</tr>
<tr>
<td>SUBPOOL</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBALIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
</tr>
<tr>
<td>GRIDLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
</tr>
<tr>
<td>ATTRAREA</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVCLASS</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>NAMELIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>LOC</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RANGE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
</tbody>
</table>

EDTINFO macro
EDTINFO macro

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

- `MF=(E, list addr)`
- `MF=(E, list addr, COMPLETE)`
- `MF=(E, list addr, NOCHECK)`

Specifies the execute form of the EDTINFO macro.

Parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>CHKGRPS</th>
<th>CHKUNIT</th>
<th>RTNUNIT</th>
<th>RTNUCBA</th>
<th>RTNGRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVNLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RECMODE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DENSITY</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTUNITL</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTDEV</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RTNATTR</th>
<th>RTNNAMD</th>
<th>RTNDEVN</th>
<th>MAXELIG</th>
<th>RTNUNAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVCOUNT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVSTAT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UNITNAME</td>
<td>UNITNAME or DEVTYPE required</td>
<td>UNITNAME or DEVTYPE required</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
<td></td>
</tr>
<tr>
<td>DEVTYPE</td>
<td>DEVTYPE or UNITNAME required</td>
<td>DEVTYPE or UNITNAME required</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
<td></td>
</tr>
<tr>
<td>SUBPOOL</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBALIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>GRIDLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>ATTRAREA</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVCLASS</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>NAMELIST</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>LOC</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RANGE</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVNLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RECMODE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>DENSITY</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTUNIT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>OUTUNIT or OUTDEV required</td>
<td>OUTUNIT or OUTDEV required</td>
</tr>
<tr>
<td>OUTDEV</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>OUTDEV or OUTUNIT required</td>
<td>OUTDEV or OUTUNIT required</td>
</tr>
</tbody>
</table>
The \textit{list addr} parameter specifies the address of the storage area for the parameter list.

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

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

\section*{EDTINFO—Modify form}

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

\section*{Syntax}

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

\begin{verbatim}
  name name: Symbol. Begin name in column 1.
  b One or more blanks must precede EDTINFO.
  EDTINFO One or more blanks must follow EDTINFO.
  b One or more blanks must follow EDTINFO.
  CHKGRPS \textbf{Note}: At least one of these functions is required: CHKGRPS,
  CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR,
  RTNAMMD, RTNDEVN, MAXELIG, RTNUNAFF. If more than
  one of these is specified, a comma must be coded between
  each of the keywords.
  CHKNUNIT \textbf{Note}: See the tables following this diagram for information on
  parameter usage with these functions.
  RTNUCBA \textbf{Note}: See the tables following this diagram for information on
  parameter usage with these functions.
  RTNGRID
  RTNATTR
  RTNAMMD
  RTNDEVN
  MAXELIG
  RTNUNAFF
  ,DEVCOUNT=devcount addr devcount addr: RS-type address or register (2) - (12).
\end{verbatim}
### EDTINFO macro

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>.DEVLIST</td>
<td>devlist addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.DEVSTAT</td>
<td>devstat addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.UNITNAME</td>
<td>unitname addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.DEVTYPE</td>
<td>devtype addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.SUBPOOL</td>
<td>subpool addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.UCBALIST</td>
<td>ucbalist addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.UCBLIST</td>
<td>ucblist addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.GRIDLIST</td>
<td>gridlist addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.ATTRAREA</td>
<td>attrarea addr: RX-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.DEVCLASS</td>
<td>devclass addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.NAMELIST</td>
<td>namelist addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.DYNAMIC</td>
<td>DYNAMIC=NO</td>
<td>Default: NO</td>
</tr>
<tr>
<td>.LOC</td>
<td>LOC=BELOW</td>
<td>Default: LOC=BELOW</td>
</tr>
<tr>
<td>.RANGE</td>
<td>RANGE=ALL</td>
<td>Default: RANGE=3DIGIT</td>
</tr>
<tr>
<td>.DEVNLIST</td>
<td>devnlist addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.RECMODE</td>
<td>recmode addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.DENSITY</td>
<td>density addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.OUTUNIT</td>
<td>outunit addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.OUTDEV</td>
<td>outdev addr: RS-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.IOCTOKEN</td>
<td>ioc-token addr: RX-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.RETCODE</td>
<td>retcode addr: RX-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.RSNCODE</td>
<td>rsncode addr: RX-type address or register (2) - (12).</td>
<td></td>
</tr>
<tr>
<td>.MF</td>
<td>(M, list addr)</td>
<td>Default: COMPLETE</td>
</tr>
<tr>
<td></td>
<td>(M, list addr, COMPLETE)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(M, list addr, NOCHECK)</td>
<td></td>
</tr>
</tbody>
</table>

The following tables show how the parameters may be specified with the CHKGPRPS, CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR, RTNNAMD, RTNDEVN, MAXELIG, and RTNUNAFF functions.
The IOCTOKEN, RETCODE, RSNCODE, and MF parameters are optional with any of the functions.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>CHKGRPS</th>
<th>CHKUNIT</th>
<th>RTNUNIT</th>
<th>RTNUCBA</th>
<th>RTNGRID</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVCOUNT</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVLIST</td>
<td>required</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVSTAT</td>
<td>optional</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UNITNAME</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVTYPE</td>
<td>not valid</td>
<td>DEVTYPE or UNITNAME required</td>
<td>required</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
</tr>
<tr>
<td>SUBPOOL</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBALIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
</tr>
<tr>
<td>GRIDLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
</tr>
<tr>
<td>ATTRAREA</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
</tr>
<tr>
<td>DEVCLASS</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>NAMELIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>LOC</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RANGE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVNLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RECMODE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DENSITY</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTUNIT</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTDEV</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RTNATTR</th>
<th>RTNNAMD</th>
<th>RTNDEVN</th>
<th>MAXELIG</th>
<th>RTNUUNAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEVCOUNT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVSTAT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UNITNAME</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
<td>UNITNAME or DEVTYPE required</td>
<td>UNITNAME or DEVTYPE required</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVTYPE</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
<td>DEVTYPE or UNITNAME required</td>
<td>DEVTYPE or UNITNAME required</td>
<td>not valid</td>
</tr>
<tr>
<td>SUBPOOL</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBALIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>UCBLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>GRIDLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>ATTRAREA</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVCLASS</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
</tbody>
</table>
EDTINFO macro

<table>
<thead>
<tr>
<th>Parameters</th>
<th>RTNATTR</th>
<th>RTNNAMD</th>
<th>RTNDEVN</th>
<th>MAXELIG</th>
<th>RTNUNAFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMELIST</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DYNAMIC</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>LOC</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>RANGE</td>
<td>not valid</td>
<td>not valid</td>
<td>optional</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>DEVNLIST</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
<td>not valid</td>
</tr>
<tr>
<td>REMODE</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>DENSITY</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>required</td>
<td>not valid</td>
</tr>
<tr>
<td>OUTUNIT</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>OUTUNIT or OUTDEV required</td>
<td>OUTUNIT or OUTDEV required</td>
</tr>
<tr>
<td>OUTDEV</td>
<td>not valid</td>
<td>not valid</td>
<td>not valid</td>
<td>OUTDEV or OUTUNIT required</td>
<td>OUTDEV or OUTUNIT required</td>
</tr>
</tbody>
</table>

Parameters

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

- `MF=(M, list addr)`
- `MF=(M, list addr, COMPLETE)`
- `MF=(M, list addr, NOCHECK)`

Specifies the modify form of the EDTINFO macro.

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

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

**Note:** When using the NOCHECK option, make sure that it is preceded by an execute or modify form invocation that specifies or defaults to the COMPLETE option. Otherwise, the parameter list might not be completely initialized.
Chapter 93. ENQ — Request control of a serially reusable resource

Description

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

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

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

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

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

Environment

The requirements for callers of ENQ are:

Minimum authorization: Problem state with any PSW key. For the SMC, ECB, TCB, MASID, and MTCB parameters or when the specified qname is ADRDFRAG, ADRDSN, ARCNQG, BWODSN, SYSZ*, SYSCTLG, SYSDSN, SYSIEA01, SYSIEECT, SYSIEFSD, SYSIGGV1, SYSIGGV2, SYSPSWRD, SYSVSAM, or SYSVTOC, the authorization must be one of the following:
- Supervisor state
- PSW key 0-7
- APF-authorized.

Dispatchable unit mode: Task
ENQ macro

Cross memory mode:

For LINKAGE=SVC: PASN=HASN=SASN

For LINKAGE=SYSTEM: Any PASN, Any HASN, Any SASN

For LINKAGE=SYSTEM with SMC=STEP: PASN=HASN, Any SASN

AMODE: 24- or 31- or 64-bit
ASC mode: Primary
Interrupt status: Enabled for I/O and external interrupts
Locks: No locks held
Control parameters: Control parameters must be in the primary address space. Except for the TCB, all parameters can reside above 16 megabytes.

Programming requirements

None.

Restrictions

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

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

The caller cannot have an EUT FRR established.

There are some considerations to be aware of when using enclaves for tasks that serialize resources using the ENQ macro. For details, see "Using ENQ/DEQ or Latch Manager Services With Enclaves" in "z/OS MVS Programming: Workload Management Services".

Input register information

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

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>One of the following:</td>
</tr>
<tr>
<td></td>
<td>• If you specify RET=TEST, RET=USE, RET=CHNG, or RET=HAVE: If all return codes for the resources named in the ENQ macro are 0, register 15 contains 0. If any of the return codes are not 0, register 15 contains the address of a storage area containing the return codes.</td>
</tr>
<tr>
<td></td>
<td>• Otherwise: Used as a work register by the system.</td>
</tr>
</tbody>
</table>
When control returns to the caller, the access registers (ARs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

**Performance implications**

None.

**Syntax**

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

```plaintext
name name

b

ENQ

b

( 

qname addr

, rname addr

, rname length

,STEP

,SYSTEM

,SYSTEMS

)
```

- `name`: symbol. Begin `name` in column 1.
- `b`: One or more blanks must precede ENQ.
- `ENQ`: One or more blanks must follow ENQ.
- `qname addr`: A-type address or register (2) - (12).
- `rname addr`: A-type address or register (2) - (12).
- `rname length`: symbol, decimal digit, or register (2) - (12).
- `STEP`: Default: STEP
- `SYSTEM`: Default: SYSTEM
- `SYSTEMS`: Default: SYSTEMS

**Note:** Code `rname length` if `rname addr` is a register.
ENQ macro

```
,RET=CHNG  Default: RET=NONE
,RET=HAVE
,RET=TEST
,RET=USE
,RET=NULL

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

,RELATED=value  value: any valid macro keyword specification.

,LINKAGE=SVC  DEFAULT: LINKAGE=SVC
,LINKAGE=SYSTEM
```

Parameters

The parameters are explained as follows:

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

  - **Note:** See [z/OS MVS Diagnosis: Reference](https://www.ibm.com/docs/en/zos) for a list of major and minor ENQ/DEQ names and the resources that issue the ENQ/DEQ.

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

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

- \( ,rname\) \( length \)
  - Specifies the length of the \( rname \). If this parameter is omitted, the system uses the assembled length of the \( rname \). To override the assembled length, specify this parameter.
ENQ macro

You can code a value between 1 and 255. Also, you can specify 0, which means that the length of the rname must be contained in the first byte at the rname addr.

STEP
SYSTEM
SYSTEMS

Specifies the scope of the resource.

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

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

SYSTEMS specifies that the resource can be shared between systems.

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

) Specifies the end of the resource description.

Notes on specifying multiple resources on one ENQ request:

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

RET=CHNG
RET=HAVE
RET=TEST
RET=USE
RET=NONE

Specifies the type of request for the resources named on the ENQ request.

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

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

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

USE control of the resources is to be assigned to the active task only if the resources are immediately available. If any of the resources are not available, the active task is not placed in a wait condition.

NONE Control of all the resources is unconditionally requested.

See "Return and reason codes" on page 544 for an explanation of the return codes for these requests.
ENQ macro

,,RNL=YES
,,RNL=NO
Controls global resource serialization RNL processing, which can cause the scope value of a resource to change. IBM recommends that you use the default, RNL=YES, to allow global resource serialization to perform RNL processing. Use RNL=NO when you are sure that you want the request to be processed only by global resource serialization using only the specified scope. When RNL=NO is specified the ENQ request will be ignored by alternative serialization products. Refer to z/OS MVS Planning: Global Resource Serialization, RNL Processing, for more information about the use of RNL=NO.

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

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

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

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

The default is LINKAGE=SVC.

ABEND codes
For only unconditional requests, the caller might encounter abend code X'138' or X'538'. For unconditional or conditional requests, the caller might encounter one of the following abend codes:
- X'238'
- X'338'
- X'438'
- X'738'
- X'838'
- X'938'

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

Return and reason codes
The system provides a return code only if you specify RET=TEST, RET=USE, RET=CHNG, or RET=HAVE; otherwise, return of the task to the active condition indicates that control of the resource has been assigned or was previously assigned to the task. If all return codes for the resources named in the ENQ macro are 0, register 15 contains 0. For nonzero return codes, register 15 contains the address of a storage area containing the return codes, as shown in Figure 6.
The return codes are placed in the parameter list resulting from the macro expansion in the same sequence as the resource names in the ENQ macro.

The return codes for the ENQ macro with the RET=TEST parameter are described in Table 15.

Table 15. Return Codes for the ENQ Macro with the RET=TEST Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning**: The resource is immediately available.  
                          | **Action**: None required. However, you might take some action based on your application. |
| 4                       | **Meaning**: The resource is not immediately available.  
                          | **Action**: None required. However, you might take some action based on your application. |
| 8                       | **Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource.  
                          | **Action**: None required. However, you might take some action based on your application.  
                          | To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; if bit 3 is on, the task has shared control. |

Figure 6. Return Code Area Used by ENQ
### Table 15. Return Codes for the ENQ Macro with the RET=TEST Parameter (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 14                      | **Meaning:** A previous request for control of the same resource has been made for the same task. The task does not have control of the resource.  
**Action:** None required. However, you might take some action based on your application. |

The return codes for the ENQ macro with the RET=USE parameter are described in Table 16.

### Table 16. Return Codes for the ENQ Macro with the RET=USE Parameter

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

The return codes for the ENQ macro with the RET=CHNG parameter are described in Table 17.

### Table 17. Return Codes for the ENQ Macro with the RET=CHNG Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | **Meaning:** The status of the resource has been changed to exclusive.  
**Action:** None. |
| 4                       | **Meaning:** The status of the resource cannot be changed to exclusive. Other tasks share the resource.  
**Action:** None required. However, you might take some action based on your application. |
Table 17. Return Codes for the ENQ Macro with the RET=CHNG Parameter (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 8                       | Meaning: The status of the resource cannot be changed to exclusive. No tasks have issued an ENQ request for the resource.  
Action: None required. However, you might take some action based on your application. |
| 14                      | Meaning: The status of the resource cannot be changed to exclusive. A previous request for control of the same resource has been made for the same task. The task does not have control of the resource.  
Action: None required. However, you might take some action based on your application. |

The return codes for the ENQ macro with the RET=HAVE parameter are described in Table 18.

Table 18. Return Codes for the ENQ Macro with the RET=HAVE Parameter

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0                       | Meaning: The active task now has control of the resource.  
Action: None. |
| 8                       | Meaning: A previous request for control of the same resource has been made for the same task. The task has control of the resource.  
Action: None required. However, you might take some action based on your application.  
To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control. |
| 14                      | Meaning: A previous request for control of the same resource has been made for the same task but that request has not yet been satisfied (such as an ENQ with RET=NONE which waits for the resource). The task does not have control of the resource.  
Action: None required. However, you might take some action based on your application. |
| 18                      | Meaning: Environmental error. The limit for the number of concurrent resource requests has been reached. The task does not have control of the resource unless some previous ENQ or RESERVE request caused the task to obtain control of the resource.  
Action: Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded. |

Example 1

Unconditionally request exclusive control of one resource and shared control of another. The system will return control to the requesting program only when both resources are available.

ENQ (MAJOR3,MINOR3,E,8,SYSTEM,MAJOR4,MINOR4,S,6,SYSTEM)
**Example 2**

Conditionally request shared control of a serially reusable resource that is known only within the address space (STEP). The resource is only to be obtained if immediately available. The resource will be used for read-only purposes. The length of \texttt{rname} is allowed to default.

\texttt{ENQ (MAJOR1,MINOR1,S,,STEP),RET=USE}

**Example 3**

Unconditionally request exclusive control of three resources. The scope of each resource differs (STEP, SYSTEM, and SYSTEMS, respectively). The \texttt{rname} length of the first resource is 3 characters and the \texttt{rname} length of the third resource is 8 characters. Allow the \texttt{rname} length of the second resource to default to its assembled length.

\texttt{ENQ (MAJOR4,MINOR4,E,3,,MAJOR2,MINOR2,,,SYSTEM, X MAJOR3,MINOR3,E,8,SYSTEMS)}

**ENQ: List form**

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

**Syntax**

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

\begin{verbatim}
name name
b ENQ
b
\end{verbatim}

\begin{verbatim}
( 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
\end{verbatim}
ENQ macro

, name_length

    name_length: symbol or decimal digit.
    Default: assembled length of name

, STEP
Default: STEP

, SYSTEM
, SYSTEMS

, RET=CHNG
    Default: RET=NONE

, RET=HAVE
, RET=TEST
, RET=USE
, RET=NONE

, RNL=YES
    Default: RNL=YES

, RNL=NO

, RELATED=value
    value: any valid macro keyword specification.

, MF=L

______________________________

Parameters

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

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

ENQ: Execute form

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

Syntax

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

name
    name: symbol. Begin name in column 1.

b
    One or more blanks must precede ENQ.

ENQ
ENQ macro

b One or more blanks must follow ENQ.

( Note: ( and ) are the beginning and end of a parameter list. The entire list is optional. If nothing in the list is desired then (, ), and all parameters between ( and ) should not be specified. If something in the list is desired, the (, ), and all parameters in the list should be specified as indicated at the left.

qname addr qname addr: RX-type address or register (2) - (12).

, rname addr rname addr: RX-type address or register (2) - (12).

,E S

, rname length rname length: symbol, decimal digit, or register (2) - (12).

,STEP,SYSTEM,SYSTEMS

) Note: See note opposite ( above.

,RET=CHNG,RET=HAVE,RET=TEST,RET=USE,RET=NONE

,RNL=YES,RNL=NO

,RELATED=value value: any valid macro keyword specification.

,LINKAGE=SVC DEFAULT: LINKAGE=SVC

,LINKAGE=SYSTEM

,MF=(E,list addr) list addr: RX-type address or register (1) - (12).

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

\texttt{,MF\=(E, list addr)}

Specifies the execute form of the ENQ macro.

\textit{list addr} specifies the area that the system uses to contain the parameters.
ENQ macro
Chapter 94. ESPIE — Extended SPIE

Description

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

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

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

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

- “Environment”
- “Programming Requirements”
- “Restrictions”
- “Performance Implications”
- “ABEND Codes”

Environment

The requirements for the caller are:

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

Dispatchable unit mode: Task

Cross memory mode: PASN=HASN=SASN

AMODE: 24- or 31- or 64-bit

ASC mode: Primary

Interrupt status: Enabled for I/O and external interrupts

Locks: No locks held

Control parameters: Must be in the primary address space.

Programming requirements

None.

Restrictions

None.

Performance implications

Programs that need to intercept only specific hardware program check interruptions (such as arithmetic exceptions or data conversion exceptions) will find ESPIE to be...
more efficient than establishing an ESTAE environment to screen all abends for specific OCx abends. This is because the operating system must do significantly more processing to enter and retry from an ESTAE routine as compared to an ESPIE routine.

ABEND codes

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

The information documented under the following headings is provided separately for each of the three options (SET, RESET, and TEST):
- “Input Register Information”
- “Output Register Information”
- “Syntax”
- “Parameters”
- “Return and Reason Codes”
- “Examples”

SET option

Input register information

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

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Token representing the previously active SPIE/ESPIE environment</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code of 0</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Syntax

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

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

b
One or more blanks must precede ESPIE.

ESPIE
b
One or more blanks must follow ESPIE.

---

**SET**

\[exit \text{addr}\]

\[\text{exit addr: A-type address, or register (2) - (12).}\]

\[,\text{interruptions}\]

\[\text{interruptions: Decimal digits 1-15 and expressed as:}\]

- single values: (2, 3, 4, 7, 8, 9, 10)
- ranges of values: ((2, 4), (7, 10))
- combinations: (2, 3, 4, 7, 10)

\[\text{,PARAM=\text{list addr}}\]

\[\text{list addr: A-type address or register (2) - (12).}\]

\[\text{,PKM=SYSTEM\_RULES}\]

\[\text{Default: PKM=SYSTEM\_RULES}\]

\[\text{,PKM=TOS}\]

---

**Parameters**

The parameters are explained as follows:

**SET**

Indicates that an ESPIE environment is to be established.

\[\text{,exit \text{addr}}\]

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

\[\text{,\text{(interruptions)}}\]

Indicates the interruption types that are being trapped. The interruption types are:

<table>
<thead>
<tr>
<th>Number</th>
<th>Interruption Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Operation</td>
</tr>
<tr>
<td>2</td>
<td>Privileged operation</td>
</tr>
<tr>
<td>3</td>
<td>Execute</td>
</tr>
<tr>
<td>4</td>
<td>Protection</td>
</tr>
<tr>
<td>5</td>
<td>Addressing</td>
</tr>
<tr>
<td>6</td>
<td>Specification</td>
</tr>
<tr>
<td>7</td>
<td>Data</td>
</tr>
<tr>
<td>8</td>
<td>Fixed-point overflow (maskable)</td>
</tr>
<tr>
<td>9</td>
<td>Fixed-point divide</td>
</tr>
<tr>
<td>10</td>
<td>Decimal overflow (maskable)</td>
</tr>
<tr>
<td>11</td>
<td>Decimal divide</td>
</tr>
<tr>
<td>12</td>
<td>Exponent overflow</td>
</tr>
<tr>
<td>13</td>
<td>Exponent underflow (maskable)</td>
</tr>
<tr>
<td>14</td>
<td>Significance (maskable)</td>
</tr>
<tr>
<td>15</td>
<td>Floating-point divide</td>
</tr>
</tbody>
</table>
ESPIE macro

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

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

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

`\,PARAM=list addr`

Specifies the fullword address of a parameter list that is to be passed by the caller to the exit routine.

`\,PKM=SYSTEM_RULES`

`\,PKM=TOS`

SYSTEM_RULES specifies that the system should determine the appropriate PSW key mask for the ESPIE exit and resume point. TOS specifies that the Time Of Set PKM should be propagated to the ESPIE exit and resume point.

**Return and reason codes**

None.

**Example**

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

```
ESPIE_SET,EXIT,(1,4),PARAM=PARMLIST
```

**ESPIE - List form**

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

**Syntax**

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

```
name

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

```
b

One or more blanks must precede ESPIE.
```

ESPIE
One or more blanks must follow ESPIE.

**SET**

(exit addr)  *exit addr*: A-type address.

**Note:** This parameter must be specified on either the list or the execute form of the macro.

(interruptions)  *interruptions*: Decimal digit 1-15 and expressed as:

- single values: (2, 3, 4, 7, 8, 9, 10)
- range of values: ((2, 4), (7, 10))
- combinations: (2, 3, 4, (7, 10))

(PARAM=list addr)  *list addr*: A-type address.

(MF=L)

**Parameters**

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

(MF=L)

Specifies the list form of the ESPIE macro.

**Example**

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

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

**ESPIE - Execute form**

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

**Syntax**

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

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

b  One or more blanks must precede ESPIE.
ESPIE macro

ESPIE

b One or more blanks must follow ESPIE.

SET

,exit addr  

exit addr: RX-type address or register (2) - (12).

Note: This parameter must be specified on either the list or the execute form of the macro.

,(interruptions)  

interruptions: Decimal digit 1-15 and expressed as:

single values:  (2, 3, 4, 7, 8, 9, 10)
range of values:  ((2, 4), (7, 10))
combinations:  (2, 3, 4, (7, 10))

,PARAM=list addr  

list addr: RX-type address or register (2) - (12).

,MF=(E,ctrl addr)  

ctrl addr: RX-type address, or register (1), (2) - (12).

,PKM=SYSTEM_RULES,PKM=TOS  

Default: PKM=SYSTEM_RULES

Parameters

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

,MF=(E,ctrl addr)

Specifies the execute form of the ESPIE macro.

Example

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

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

RESET option

The RESET option of the ESPIE macro cancels the current SPIE/ESPIE environment and re-establishes the previously active SPIE/ESPIE environment identified by the token specified.

Input register information

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

Output register information

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

**Syntax**

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

```
  name

  b

  ESPIE

  b

  RESET

  ,token
```

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

*b*  
One or more blanks must precede ESPIE.

**Parameters**

The parameters are explained as follows:

**RESET**

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

*,token*

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

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

Return and reason codes
None.

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

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

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

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>1-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

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

```
{name} {name: Symbol. Begin name in column 1.}
b One or more blanks must precede ESPIE.
ESPIE
b One or more blanks must follow ESPIE.
```
Parameters

The parameters are explained as follows:

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

*,parm addr*
Specifies the address of a 4-word parameter list aligned on a fullword boundary.

The parameter list has the following form:

<table>
<thead>
<tr>
<th>Word</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Address of the exit routine (31-bit address with the high-order bit set to 0 for 24-bit routines or 1 for 31-bit routines)</td>
</tr>
<tr>
<td>1</td>
<td>Address of the user-defined parameter list</td>
</tr>
<tr>
<td>2</td>
<td>Mask of program interruption types</td>
</tr>
<tr>
<td>3</td>
<td>Zero</td>
</tr>
</tbody>
</table>

Return and reason codes

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

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><strong>Meaning:</strong> An ESPIE exit is active and the 4-word parameter list contains the information specified in the description of the <em>parm addr</em> parameter.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Meaning:</strong> A SPIE exit is active. Word 1 of the parameter list described under <em>parm addr</em> contains the address of the current PICA. Words 0, 2, and 3 of the parameter list contain no relevant information.</td>
</tr>
<tr>
<td>8</td>
<td><strong>Meaning:</strong> No SPIE or ESPIE is active. The contents of the 4-word parameter list contain no relevant information.</td>
</tr>
</tbody>
</table>

Example

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

**ESPIE TEST,PARMLIST**
ESPIE macro
Chapter 95. ESTAE and ESTAEX — Extended specify task abnormal exit

Description

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

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

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

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

- If your program is to execute in 31-bit addressing mode, you must use the SP Version 2 of the ESTAE macro or a later version. For information about how to select a macro for an MVS/SP version other than the current version, see "Compatibility of MVS macros" on page 1.
- Callers that are in primary address space control (ASC) mode and not in cross-memory mode can issue either ESTAE or ESTAEX.
- Callers that are in access register (AR) mode or in cross-memory mode must use ESTAEX.
- IBM recommends that all callers use the ESTAEX macro, unless your program and your recovery routine are in 24-bit addressing mode, in which case you should use ESTAE.

Depending on whether you code ESTAE or ESTAEX, the system passes the address of the user-specified parameter area differently. The SDWAPARM field in the SDWA contains either the address of the parameter area (ESTAE), or the address of a doubleword that contains the address and ALET of the parameter area (ESTAEX). When you run in AMODE 64 (as indicated by specifying AMODE64=YES via the SYSSTATE macro) and invoke ESTAEX, your ESTAEX routine will get control in AMODE 64. The 8-byte area pointed to by the SDWAPARM field will be the 8-byte address of the parameter area. Note that no ALET information is provided to the ESTAEX routine in this case.

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

The descriptions of ESTAE and ESTAEX in this book are:
ESTAE and ESTAEX macros

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

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

### Environment
The requirements for the caller are:

- **Minimum authorization:** Problem state and any PSW key.
- **Dispatchable unit mode:** Task
- **Cross memory mode:** PASN=HASN=SASN
- **AMODE:** 24- or 31-bit for ESTAE; 24- or 31- or 64-bit for ESTAEX
- **ASC mode:** Primary
- **Interrupt status:** Enabled for I/O and external interrupts
- **Locks:** No locks held
- **Control parameters:** Must be in the primary address space.

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

### Restrictions
- For SVC-entry, you must have no EUT FRRs.
- For branch entry, IBM recommends that you have no EUT FRRs.

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

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code if GPR 15 contains X'4'; otherwise, used as a work register by the system</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
</tbody>
</table>
15 Return code

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

**Performance implications**
None.

**Syntax**

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

```
name name

\slash

ESTAE
\slash

exit addr exit addr

0 ,CT

,OV

,PARAM=list addr

,XCTL=NO

,XCTL=YES

,PURGE=NONE

,PURGE=QUIESCE

,PURGE=HALT

,ASYNCH=YES

,ASYNCH=NO

,TERM=NO

,TERM=YES

,RELATED=value
```

A-type address, or register (2) - (12).

Default: CT

Default: XCTL=NO

Default: PURGE=NONE

Default: ASYNCH=YES

Default: TERM=NO

value: Any valid macro keyword specification.
ESTAE and ESTAEX macros

Parameters

The parameters are explained as follows:

\texttt{exit addr}

- \texttt{0} Specifies the 31-bit address of an ESTAE recovery routine to be entered if the task issuing this macro ends abnormally. If 0 is specified, the most recent ESTAE recovery routine is deactivated and no longer defined.

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

\texttt{,CT}

\texttt{,OV}

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

\texttt{,PARAM=list addr}

- Specifies the address of a user-defined parameter area containing data to be used by the ESTAE recovery routine when it is scheduled for execution.

\texttt{,XCTL=NO}

\texttt{,XCTL=YES}

- Specifies that the ESTAE recovery routine will be deactivated and no longer defined (NO) or will remain activated and defined (YES) if an XCTL macro is issued by this program.

\texttt{,PURGE=NONE}
\texttt{,PURGE=QUIESCE}
\texttt{,PURGE=HALT}

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

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

If PURGE=NONE is specified, all data areas affected by input/output processing may continue to change during ESTAE recovery routine processing.

If PURGE=NONE is specified and the error was an error in input/output processing, recursion will develop when an input/output interruption occurs,
even if the recovery routine is in progress. Thus, it will appear that the recovery routine failed when, in reality, input/output processing was the cause of the failure.

Do not use PURGE=HALT to stop processing a data set if you expect to continue reading the data set at a different point.

**Notes:**

1. You need to understand PURGE processing before using this parameter. For information about PURGE processing, see [z/OS DFSMSdfp Advanced Services](http://www.ibm.com/support/docview.wss?uid=swg27007321).
2. When using PURGE, you should consider any access-method ramifications. See the appropriate DFP manual for the particular access method you are using to determine these ramifications.
3. The system performs the requested I/O processing only for the first ESTAE-type recovery routine that gets control. Subsequent routines that get control receive an indication of the I/O processing previously done, but no additional processing is performed.

\*ASYNCH=YES\*

\*ASYNCH=NO\*

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

ASYNCH=YES must be coded if:

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

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

\*TERM=NO\*

\*TERM=YES\*

Specifies that the recovery routine associated with the ESTAE request will be scheduled (YES) or will not be scheduled (NO), in addition to normal ESTAE processing, in the following situations:

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

When the recovery routine is entered because of one of the preceding reasons, retry will not be permitted. If a dump is requested at the time the ABEND macro is issued, it is taken prior to entry into the recovery routines.

**Note:** If DETACH was issued with the STAE parameter, the following will occur for the task to be detached:
- All ESTAE recovery routines will be entered.
- The most recently activated STAE recovery routine will be entered.
- All STAI/ESTAI recovery routines will be entered unless return code 16 is returned from one of the STAI recovery routines.

In these cases, entry to the recovery routine is prior to dumping and retry will not be permitted.

\[\text{RELATED}=value\]
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.

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

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

ABEND codes

X'13C'
See [z/OS MVS System Codes](#) for an explanation and programmer response for the abend code.

Return and reason codes

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

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

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 04                      | 04                      | **Meaning:** Program error. ESTAE OV was specified but ESTAE CT was performed. The last ESTAE recovery routine was not owned by the user's RB.  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 04                      | 08                      | **Meaning:** Program error. ESTAE OV was specified but ESTAE CT was performed. The last ESTAE recovery routine was not created at the current linkage stack level.  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 04                      | 0C                      | **Meaning:** Program error. ESTAE OV was specified but ESTAE CT was performed. The last recovery routine was not an ESTAE recovery routine.  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 0C                      | none                    | **Meaning:** Program error. A recovery routine address equal to zero was specified and either there are no recovery routines for this task, the most recent recovery routine is not owned by the caller, or the most recent recovery routine is not an ESTAE recovery routine.  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 10                      | none                    | **Meaning:** System error. An unexpected error was encountered while this request was being processed.  
**Action:** Rerun your program one or more times. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel. |
| 14                      | none                    | **Meaning:** System error. ESTAE was unable to obtain storage for a system data area.  
**Action:** Rerun your program one or more times. If the problem persists, check with the operator to see if the installation is experiencing a storage constraint problem. |
| 18                      | none                    | **Meaning:** Program error. ESTAE OV was specified without the TOKEN parameter, but the ESTAE recovery routine was created with the TOKEN parameter. (The TOKEN parameter is available only to programs in supervisor state with PSW key 0-7 or programs that are APF-authorized.)  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 1C                      | none                    | **Meaning:** Program error. ESTAE was unable to access the input parameter area.  
**Action:** Make sure the parameter area is in the primary address space and reissue the ESTAE macro. |
| 20                      | none                    | **Meaning:** Program error. XCTL= YES was rejected because the linkage stack was not at the same level as it was when the RB was created.  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
ESTAE and ESTAEX macros

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 24                      | none                    | **Meaning:** Program error. A recovery routine address equal to zero was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level.  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |
| 28                      | none                    | **Meaning:** Program error. ESTAE OV was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level.  
**Action:** Correct the environment and either reissue the ESTAE macro or rerun your program, as appropriate. |

**Example 1**

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

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

**Example 2**

Provide the pointer to the recovery code in the register called EXITPTR, and the address of the ESTAE recovery routine parameter area in register 9. Register 8 points to the area where the ESTAE parameter area (created with the MF=L option) is to be modified.

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

**ESTAEX —Extended specify task abnormal exit**

**Note**

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

**Environment**

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

- **Cross memory mode:** Any PASN, any HASN, any SASN
- **ASC mode:** Primary or access register (AR)

**Programming requirements**

If the program is in AR mode:

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

The caller of ESTAEX cannot have an EUT FRR established.

Syntax

The parameters on the standard form of the ESTAEX macro are exactly the same as for the standard form of the ESTAE macro, except that the SDWALOC31 parameter is available only on the ESTAE macro. The SDWA is always placed in 31-bit storage for an ESTAEX recovery routine, so the parameter is unnecessary for ESTAEX. They are written as follows:

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

b
One or more blanks must precede ESTAEX.

ESTAEX

b
One or more blanks must follow ESTAEX.

exit addr
exit addr: A-type address, or register (2) - (12).

0

,CT
Default: CT

,OV

,PARAM=list addr
list addr: A-type address, or register (2) - (12).

,XCTL=NO
Default: XCTL=NO

,XCTL=YES

,PURGE=NONE
Default: PURGE=NONE

,PURGE=QUIESCE

,PURGE=HALT

,ASYNCH=YES
Default: ASYNCH=YES

,ASYNCH=NO

,TERM=NO
Default: TERM=NO

,TERM=YES

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

Parameters

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

ABEND codes

X'13C'

See [z/OS MVS System Codes](#) for an explanation and programmer response for the
abend code.

Return and reason codes

When control is returned to the instruction following the ESTAEX macro, the return
code in GPR 15 and the reason code in GPR 0 might be different from those for
the ESTAE macro. The return and reason codes for ESTAEX are listed below.

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00                       | none                    | **Meaning:** Successful completion of ESTAEX request.  
|                          |                         | **Action:** None. |
| 04                       | 00                      | **Meaning:** Program error. ESTAEX OV was specified but ESTAEX CT was performed. No valid ESTAE recovery routine existed.  
|                          |                         | **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 04                       | 04                      | **Meaning:** Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last ESTAE recovery routine was not owned by the user's RB.  
|                          |                         | **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 04                       | 08                      | **Meaning:** Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last ESTAE recovery routine was not owned by the user's linkage stack entry.  
|                          |                         | **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 04                       | 0C                      | **Meaning:** Program error. ESTAEX OV was specified but ESTAEX CT was performed. The last recovery routine was not an ESTAE recovery routine.  
|                          |                         | **Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 08                       | none                    | **Meaning:** Program error. The ESTAEX request was not valid.  
|                          |                         | **Action:** Correct the request and either reissue the ESTAEX macro or rerun your program, as appropriate. |
### ESTAE and ESTAEX macros

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Hexadecimal Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 0C                      | none                    | **Meaning:** Program error. A recovery routine address equal to zero was specified and either there are no recovery routines for this TCB, the most recent recovery routine is not owned by the caller, or the most recent recovery routine is not an ESTAE recovery routine.  
**Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 10                      | none                    | **Meaning:** System error. An unexpected error was encountered while the request was being processed.  
**Action:** Rerun your program one or more times. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel. |
| 14                      | none                    | **Meaning:** System error. ESTAEX was unable to obtain storage for a system data area.  
**Action:** Rerun your program one or more times. If the problem persists, check with the operator to see if the installation is experiencing a storage constraint problem. |
| 18                      | none                    | **Meaning:** Program error. ESTAEX OV was specified without the TOKEN parameter, but the ESTAE recovery routine was created with the TOKEN parameter. (The TOKEN parameter is available only to programs in supervisor state with PSW key 0-7 or programs that are APF-authorized.)  
**Action:** Correct the environment and either reissue the ESTAEX macro or rerun your program, as appropriate. |
| 1C                      | none                    | **Meaning:** Program error. ESTAEX was unable to access the input parameter area.  
**Action:** Make sure the parameter area is contained in the primary address space and reissue the ESTAEX macro or rerun your program, as appropriate. |
| 20                      | none                    | **Meaning:** Program error. XCTL=YES was rejected because the linkage stack was not at the same level as it was when the RB was created.  
**Action:** Correct the environment and reissue the ESTAEX macro or rerun your program, as appropriate. |
| 24                      | none                    | **Meaning:** Program error. A recovery routine address equal to zero was specified, but it was rejected because no ESTAE recovery routines were active for the current linkage stack level.  
**Action:** Correct the environment and reissue the ESTAEX macro or rerun your program, as appropriate. |
ESTAE and ESTAEX macros

ESTAE and ESTAEX—List form

The list form of the ESTAE and ESTAEX macros is used to construct a remote control parameter area.

Syntax

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

```
name name

bslash

ESTAE

bslash

ESTAEX

b

One or more blanks must precede ESTAE or ESTAEX.

exit addr exit addr

0

,PARAM=list addr

,ASYNCH=YES,ASYNCH=NO

,TERM=NO,TERM=YES

,RELATED=value

,SDWALOC31=NO,SDWALOC31=YES

,MF=L
```

name: Symbol. Begin name in column 1.

b: One or more blanks must follow ESTAE or ESTAEX.

exit addr: A-type address.

list addr: A-type address.

Default: PURGE=None

Default: ASYNCH=YES

Default: TERM=NO

value: Any valid macro keyword specification.

Default: SDWALOC31=NO

Note: SDWALOC31 is supported only by ESTAE.

Parameters

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

, MF=L

Specifies the list form of ESTAE or ESTAEX.
ESTAE and ESTAEX—Execute form

A remote control parameter area is used in, and can be modified by, the execute form of the ESTAE and ESTAEX macros. The control parameter area can be generated by the list form of ESTAE or ESTAEX. A user who wants to dynamically change the contents of the remote control parameter area can code a new recovery routine address (exit addr) or a new parameter area address (PARAM). If exit addr or PARAM is coded, only the associated field in the remote control parameter area will be changed. The other fields will remain as they were before the current ESTAE or ESTAEX request was made.

Syntax

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

```
name

name: Symbol. Begin name in column 1.

b

One or more blanks must precede ESTAE or ESTAEX.

ESTAE

ESTAEX

b

One or more blanks must follow ESTAE or ESTAEX.

exit addr

exit addr: RX-type address, or register (2) - (12).

0

,CT

,CV

,PARAM=list addr

list addr: RX-type address, or register (2) - (12).

,XCTL=NO

,XCTL=YES

,PURGE=NONE

,PURGE=QUIESCE

,PURGE=HALT

,ASYNCH=YES

,ASYNCH=NO

,TERM=NO

,TERM=YES

,RELATED=value

value: Any valid macro keyword specification.

,SDWALOC31=NO

Default: SDWALOC31=NO

Note: SDWALOC31 is supported only by ESTAE.

,SDWALOC31=YES

,MF=(E,ctrl addr)

ctrl addr: RX-type address, or register (1) or (2) - (12).
```
ESTAE and ESTAEX macros

Parameters

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

\[ \text{MF=\{E,ctrl addr\}} \]

Specifies the execute form of the ESTAE and ESTAEX macro using a remote control parameter area.
Chapter 96. EVENTS — Wait for one or more events to complete

Description

The EVENTS macro is the same as the WAIT macro with the ECBLIST parameter, with one additional function: EVENTS notifies the calling program that event control blocks (ECBs) have completed and the order in which they completed.

The macro performs the following functions:
- Creates and deletes EVENTS tables.
- Initializes and maintains a list of completed event control blocks.
- Provides for single or multiple ECB processing.

For a detailed explanation of how to use EVENTS to perform these functions see "Using the EVENTS macro" on page 579.

If your program is to execute in 31-bit addressing mode, you must use the SP Version 2 expansion of this macro or a later version. For information about how to select the macro for an MVS/SP version other than the current version, see "Compatibility of MVS macros" on page 1.

Environment

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 primary address space.

Programming requirements

None.

Restrictions

None.

Input register information

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

Output register information

When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>
EVENTS macro

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

Performance implications

None.

Syntax

The EVENTS macro is written as follows:

```assembly
name name

b

EVENTS

b
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRIES=n</td>
<td>n: Variable or decimal digit 1-32,767.</td>
</tr>
<tr>
<td>ENTRIES=addr</td>
<td>addr: Register (2) - (12).</td>
</tr>
<tr>
<td>ENTRIES=DEL,TABLE=table address</td>
<td>Note: If ENTRIES=n or ENTRIES=DEL,TABLE=table address is specified, no other parameter should be specified.</td>
</tr>
<tr>
<td>TABLE=table address</td>
<td>table address: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,WAIT=NO</td>
<td>Default: None.</td>
</tr>
<tr>
<td>,WAIT=YES</td>
<td></td>
</tr>
<tr>
<td>,ECB=ecb address</td>
<td>ecb address: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
<tr>
<td>,LAST=last address</td>
<td>last address: Symbol, RX-type address, or register (2) - (12).</td>
</tr>
</tbody>
</table>

Note: Optional parameters are only valid when TABLE=table address is the only required parameter specified.

Parameters

The parameters are explained as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENTRIES=n</td>
<td>Specifies either a register or a decimal number from 1 to 32,767 that specifies the maximum number of completed ECB addresses that can be processed in an EVENTS table concurrently.</td>
</tr>
</tbody>
</table>
EVENTS macro

**Note:** When this parameter is specified no other parameter should be specified.

**ENTRIES=DEL, TABLE=table address**

Specifies that the EVENTS table whose address is specified by TABLE=table address is to be deleted. The user is responsible for deleting all of the tables he creates; however, all existing tables are automatically freed at task termination.

**Notes:**
1. When this parameter is specified no other parameter should be specified.
2. table address specifies a storage location below 16 megabytes.

**TABLE=table address**

Specifies either a register number or the address of a word containing the address of the EVENTS table associated with the request. The address specified with the operand TABLE must be that of an EVENTS table created by this task.

**Note:** table address specifies a storage location below 16 megabytes.

**,WAIT=NO, WAIT=YES**

Specifies whether or not to put the issuing program in a wait state when there are no completed events in the EVENTS table (specified by the TABLE= parameter).

**,ECB=ecb address**

Specifies either a register number or the address of a word containing the address of an event control block. The EVENTS macro should be used to initialize any event-type ECB. To avoid the accidental destruction of bit settings by a system service such as an access method, the ECB should be initialized after the system service that will post the ECB has been initiated (thus making the ECB eligible for posting) and before the EVENTS macro is issued to wait on the EVENTS table.

**Notes:**
1. Register 1 should not be specified for the ECB address.
2. This parameter may not be specified with the LAST= parameter.
3. If only ECB initialization is being requested, neither WAIT=NO nor WAIT=YES should be specified, to prevent any unnecessary WAIT processing from occurring.

**,LAST=last address**

Specifies either a register number or the address of a word containing the address of the last EVENT parameter list entry processed.

**Notes:**
1. Register 1 should not be specified for the LAST address.
2. This parameter should not be specified with the ECB= parameter.
3. last address specifies a storage location below 16 megabytes.

**Using the EVENTS macro**

The following explains the different uses of EVENTS:

- Creating EVENTS Tables — When ENTRIES=n is specified, the system creates an EVENTS table with “n” entries for completed ECB addresses. This table is queued on the EVENTS table queue associated with the task. (There is no limit to the number of EVENTS tables that can be queued for a single task.) The address of the EVENTS table is returned to the user in register 1. See Figure 7.
EVENTS macro

- Deleting EVENTS Tables — When ENTRIES=DEL, TABLE=table address is specified, the EVENTS table whose address is specified by the TABLE=table address parameter shall be deleted. The address specified with the TABLE operand must be that of an EVENTS table created by this task. The user is responsible for deleting all of the tables he creates; however, all existing tables are automatically freed at task termination.

- Initializing ECBs — When an ECB is created, bits 0 (wait bit) and bit 1 (post bit) must be set to zero. When an EVENTS ECB= macro is issued, bit 0 of the associated event control block is set to 1. When a POST macro is issued, bit 1 of the associated event control block is set to 1 and bit 0 is set to 0. If the ECB is reused, bit 0 and bit 1 must be set to zero before either a WAIT, EVENTS ECB=, or POST macro can be specified. If, however, the bits are set to zero before the ECB has been posted, any task waiting for that ECB to be posted will remain in wait state.

- Maintaining a List of Completed EVENT Control Blocks — After the ECB has been initialized, the POST macro sets the complete bit and puts the address of the completed ECB in the EVENTS table.

- Providing Single or Multiple ECB Processing — When the WAIT parameter is specified and there are completed ECBs in the EVENTS table, the address of the parameter list is returned in register 1. The parameter list has the following format:

Figure 7. Creating a Table

- Deleting EVENTS Tables — When ENTRIES=DEL, TABLE=table address is specified, the EVENTS table whose address is specified by the TABLE=table address parameter shall be deleted. The address specified with the TABLE operand must be that of an EVENTS table created by this task. The user is responsible for deleting all of the tables he creates; however, all existing tables are automatically freed at task termination.

- Initializing ECBs — When an ECB is created, bits 0 (wait bit) and bit 1 (post bit) must be set to zero. When an EVENTS ECB= macro is issued, bit 0 of the associated event control block is set to 1. When a POST macro is issued, bit 1 of the associated event control block is set to 1 and bit 0 is set to 0. If the ECB is reused, bit 0 and bit 1 must be set to zero before either a WAIT, EVENTS ECB=, or POST macro can be specified. If, however, the bits are set to zero before the ECB has been posted, any task waiting for that ECB to be posted will remain in wait state.

- Maintaining a List of Completed EVENT Control Blocks — After the ECB has been initialized, the POST macro sets the complete bit and puts the address of the completed ECB in the EVENTS table.

- Providing Single or Multiple ECB Processing — When the WAIT parameter is specified and there are completed ECBs in the EVENTS table, the address of the parameter list is returned in register 1. The parameter list has the following format:
The parameter list contains completed ECB addresses in post occurrence order. The high order bit of the last word in the list is set to 1. Note that the returned list can change dynamically if additional ECBs are posted while the user is processing the ECBs in the returned list. For each additional ECB that is posted, the address of the posted ECB with the high order bit set to 1 is appended after the last ECB in the table, and the high order bit of the entry before the new entry is reset to 0. The user may choose to process the entire list (see LAST parameter) or one event at a time by successive EVENTS requests with the WAIT= option.

However, if WAIT=NO is specified and no ECBs are posted in the EVENTS table, register 1 contains a zero when the user receives control.

When a user has processed more than one ECB in the parameter list returned from the previous EVENTS WAIT= macro, the LAST= parameter should be used to indicate the last ECB processed. The EVENTS macro removes from the parameter list all entries from the first thru the last specified by LAST, and then completes processing the request according to the WAIT= specification.

In the illustration that follows, ECBs 6 - 10 are posted to the parameter list after the user processed the list containing ECBs 1 - 5 and has issued another EVENTS WAIT= macro.
In the illustration that follows, ECBs 6 through 10 were posted to the parameter list, which changes dynamically while the user is processing ECBs 1 through 5.
After an ECB is posted, take the following steps:
1. Call EVENTS TABLE=,LAST= to mark the ECB as processed.
2. Clear both the wait and post bits in the ECB.
3. Read the ECB to the events table through an EVENTS TABLE=,ECB= call.

**ABEND codes**

The caller might encounter one of the following ABEND codes:
- X'17A'
- X'17D'
- X'37A'
EVENTS macro

- X'37D'
- X'47A'
- X'47D'
- X'57D'
- X'67D'
- X'77D'
- X'87D'

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

Return and reason codes
None.

Example 1

The following shows total processing via EVENTS.

EVENTS and ECB Initialization:

```
START
EVENTS ENTRIES=1000
ST R1,TABADD
WRITE ECBA
LA R2,ECBA
EVENTS TABLE=TABADD,ECB=(R2)
```

Parameter List Processing:

```
BEGIN
EVENTS TABLE=TABADD,WAIT=YES
LR R3,R1 PARMLIST ADDR
B LOOP2 GO TO PROCESS ECB
LOOP1 EVENTS TABLE=TABADD,WAIT=YES,LAST=(R3)
LR R3,R1 SAVE POINTER
LOOP2 EQU * PROCESS COMPLETED EVENTS
TM 0(R3),X'80' TEST FOR MORE EVENTS
BO LOOP1 IF NONE, GO WAIT
LA R3,4(R3) GET NEXT ENTRY
B LOOP2 GO PROCESS NEXT ENTRY
```

Deleting EVENTS Table:

```
EVENTS TABLE=TABADD,ENTRIES=DEL
TABADD DS F
```

Example 2

Processing One ECB at a Time:

```
EVENTS ENTRIES=10
ST 1,TABLE
NEXTREC GET TPDATA,KEY
ENQ (RESOURCE,ELEMENT,E,,SYSTEM)
READ DECBRW,KU,,S',MF=E
LA 3,DECBRW
EVENTS TABLE=TABLE,ECB=(3),WAIT=YES
WRITE DECBRW,K, MF=E
LA 3,DECBRW
RETEST EVENTS TABLE=TABLE,ECB=(3),WAIT=NO
LTR 1,1
BNZ NEXTREC
B RETEST
TABLE DS F
```
Chapter 97. FREEMAIN — Free virtual storage

Description

Use the FREEMAIN macro to free one or more areas of virtual storage. You can also use the FREEMAIN macro to free an entire virtual storage subpool if it is owned by the task under which your program is issuing the FREEMAIN. For more information on releasing a subpool, see the chapter about virtual storage management in z/OS MVS Programming: Assembler Services Guide.

You can also use the STORAGE macro to free storage, even if the storage was obtained using the GETMAIN macro. Compared to FREEMAIN, STORAGE provides an easier-to-use interface and has no restrictions. If your program is running in AR-mode or cross-memory mode, use the STORAGE macro to free storage.

Environment

The requirements for the caller are:


For subpools 131 and 132: a PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be released.

Dispatchable unit mode: Task.

Cross memory mode: PASN=HASN=SASN.

AMODE: 24- or 31-bit.

• For RU, RC requests: The system treats all addresses and values as 31-bit.

• For all other requests: If the calling program is in 31-bit mode, the system treats all addresses and values, passed to the FREEMAIN macro, as 31-bit. Otherwise, the system treats addresses and values as 24-bit.

ASC mode: Primary.

Interrupt status: Enabled for I/O and external interrupts.

Locks: No locks held.

Control parameters: For LC, LU, L, VC, VU, V, EC, EU, E requests: control parameters must be in the primary address space. For other requests: control parameters are in registers.

Programming requirements

None.

Restrictions

• Parameters passed to the FREEMAIN macro must not reside within the area being freed. If this restriction is violated and the parameters are the last allocated areas on a virtual page, the whole page is freed and FREEMAIN ends abnormally with an X'0C4' abend code.

• The current task ends abnormally if the specified virtual storage area does not start on a doubleword boundary or, for an unconditional request, if the specified area or subpool is not owned by the task identified as the owner of the storage.

• For SVC entry, the caller cannot have an EUT FRR established.
**FREEMAIN macro**

**Input register information**
Before issuing the FREEMAIN 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:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>For a conditional request, contains the return code. For an unconditional request, used as a work register by the system.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

**Performance implications**
None.

**Syntax**
The standard form of the FREEMAIN macro is written as follows:

```assembly
name name

name: symbol. Begin name in column 1.

b
One or more blanks must precede FREEMAIN.

FREEMAIN

b
One or more blanks must follow FREEMAIN.

LC,LA=length addr
LU,LA=length addr
L,LA=length addr
VC
VU
V
EC,LV=length value
EU,LV=length value

length addr: A-type address, or register (2) - (12).
length value: symbol, decimal number, or register (2) - (12).```
FREEMAIN macro

E,LV=length value
RC,LV=length value
RC,SP=subpool nmbr

If R, RC, or RU is specified, register (0) may also be used.

subpool nmbr: symbol, decimal number 0-127, 131, 132, or register (2) - (12).
If R is specified, register (0) may also be used.

Note: For a subpool release (RC,SP or RU,SP, or R,SP), no other
parameters except RELATED may be specified.

RU,LV=length value
RU,SP=subpool nmbr
R,LV=length value
R,SP=subpool nmbr

,A=addr

addr: A-type address, or register (2) - (12). If R, RC, or RU is specified,
register (1) can also be used.

Note: If R, RC, or RU is specified, register (1) can also be specified.

,SP=subpool nmbr

subpool nmbr: symbol, decimal number 0-127, 131, 132, or register (2) - (12).
Default: SP=0. If R is specified, register (0) may also be used.

,KEY=number

nmbr: decimal numbers 0-15, or register (2) - (12).
Note: KEY may be specified only with RC or RU.

,RELATED=value

value: any valid assembler character string.

Parameters

The parameters are explained as follows:

LC,LA=length addr
LU,LA=length addr
L,LA=length addr
VC
VU
V
EC,LV=length value
EU,LV=length value
E,LV=length value
RC,LV=length value
RC,SP=subpool nmbr
RU,LV=length value
RU,SP=subpool nmbr
R,LV=length value
R,SP=subpool nmbr

Specifies the type of FREEMAIN request:

LC, LU, and L indicate conditional (LC) and unconditional (LU and L) list
requests and specify release of one or more areas of virtual storage. The length
of each virtual storage area is indicated by the values in a list beginning at the
address specified in the LA parameter. The address of each of the virtual
storage areas must be provided in a corresponding list whose address is
specified in the A parameter. All virtual storage areas must start on a
doubleword boundary.
VC, VU, and V indicate conditional (VC) and unconditional (VU and V) variable requests and specify release of single areas of virtual storage. The address and length of the virtual storage area are provided at the address specified in the A parameter.

EC, EU, and E indicate conditional (EC) and unconditional (EU and E) element requests and specify release of single areas of virtual storage. The length of the single virtual storage area is indicated in the LV parameter. The address of the virtual storage area is provided at the address indicated in the A parameter.

RC, RU, and R indicate conditional (RC) and unconditional (RU and R) register requests and specify either the release of all the storage in a subpool or the release of a certain area in a subpool. For information on how to release all the storage in a subpool, see the description for the SP parameter. If the release is for a certain area in a subpool, the address of the virtual storage area is indicated in the A parameter. The length of the area is indicated in the LV parameter. The virtual storage area must start on a doubleword boundary.

Notes:
1. For a conditional request, errors detected while processing a FREEMAIN request with incorrect or inconsistent parameters cause the FREEMAIN service to return to the caller with a non-zero return code. For all other errors, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.
   For an unconditional request, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.
2. If the address of the area to be freed is above 16 megabytes, you must use RC or RU.

LA specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. One word is required for each virtual storage area to be released; the high-order bit in the last word must be set to 1 to indicate the end of the list. Each word must contain the required length in the low-order three bytes. The fullwords in this list must correspond with the fullwords in the associated list specified in the A parameter. The words must not be in the area to be released. If this rule is violated and if the words are the last allocated items on a virtual page, the whole page is returned to storage and the FREEMAIN abends with an X'0C4' abend code.

LV specifies the length, in bytes, of the virtual storage area being released. The value should be a multiple of 8; if it is not, the control program uses the next high multiple of 8.
   • If you specify R,LV=(0) you cannot specify the SP parameter. You must specify the subpool in register 0; the high-order byte must contain the subpool number and the low-order three bytes must contain the length unless you are requesting a subpool release. On a subpool release, the low-order three bytes must contain zeros.
   • If you specify R,LV using a symbol, decimal number, or register 2-12, you can specify the SP parameter using registers 0 or 2-12.

A=addr
Specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary.
   • If E, EC, or EU is coded, one word is required, which contains the address of the virtual storage area to be released.
   • If V, VC, or VU is coded, two words are required; the first word contains the address of the virtual storage area to be released, and the second word contains the length of the area to be released.
If L, LC, or LU is coded, one word is required for each virtual storage area to be released; each word contains the address of one virtual storage area.

If R, RC, or RU is coded, one word is required, which contains the address of the virtual storage area to be released. If R, RC, or RU is coded and *addr* specifies a register, register 1 through 12 can be used and must contain the address of the virtual storage area to be released.

Do not specify a storage address of 0 with a storage length of 0. This combination causes FREEMAIN to free the subpool specified with the SP parameter, or subpool 0 if the SP parameter is omitted.

**,SP=*

Specifies the subpool number of the virtual area to be released. Valid subpools numbers are 0-127, 131, and 132. The SP parameter is optional and if omitted, subpool 0 is assumed. If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero.

A request to release all the storage in a subpool is known as a subpool release. To issue a subpool release, specify RC,SP or RU,SP or R,SP, and do not use the A or the KEY parameter. The following subpools are valid on the SP parameter for a subpool release: 0-127, 131, and 132. An attempt to issue a subpool release for any other subpool causes an abend X'478' or X'40A'. For information about subpools, see z/OS MVS Programming: Assembler Services Guide.

**,KEY=*

Specifies the storage key in which the storage was obtained. The valid storage keys are 0-15. If a register is specified, the storage key must be in bits 24-27 of the register. KEY can be specified for subpools 131 and 132.

**,RELATED=value**

Specifies information used to self-document macros by “relating” functions or services to corresponding functions or services and can be any valid assembler character string.

**ABEND codes**

Abend codes FREEMAIN might issue are listed below in hexadecimal. For detailed abend code information, see z/OS MVS System Codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
</tbody>
</table>

**Return and reason codes**

When the FREEMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

Table 19. Return Codes for the FREEMAIN Macro

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Meaning: Successful completion. Action: None.</td>
</tr>
</tbody>
</table>
Table 19. Return Codes for the FREEMAIN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td><strong>Meaning</strong>: Program error. Not all requested virtual storage was freed.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Check your program for the following kinds of errors:</td>
</tr>
<tr>
<td></td>
<td>• The address of the storage area to be freed is not correct.</td>
</tr>
<tr>
<td></td>
<td>• The subpool you have specified does not match the subpool of the storage to be freed.</td>
</tr>
<tr>
<td></td>
<td>• The key you have specified does not match the key of the storage to be freed.</td>
</tr>
<tr>
<td>8</td>
<td><strong>Meaning</strong>: Program error. No virtual storage was freed because part of the storage area to be freed is fixed.</td>
</tr>
<tr>
<td></td>
<td><strong>Action</strong>: Determine whether you have made one of the following errors. If so, correct your program and rerun it:</td>
</tr>
<tr>
<td></td>
<td>• You passed an incorrect storage area address to the FREEMAIN macro.</td>
</tr>
<tr>
<td></td>
<td>• You attempted to free storage that is fixed.</td>
</tr>
</tbody>
</table>

**Example 1**

Free 400 bytes of storage from subpool 10. Register 1 contains the address of the storage area. If the storage is not allocated to the current task, do not abnormally terminate the caller.

FREEMAIN RC, LV=400, A=(1), SP=10

**Example 2**

Free all of subpool 3 (if any) that belongs to the current task. If the request is not successful, abnormally terminate the caller.

FREEMAIN RU, SP=3

**Example 3**

Free from subpool 5, three areas of storage of 200, 800, and 32 bytes, previously obtained using the list and execute forms of the GETMAIN macro. Storage area addresses are in AREAADD. If any of the storage areas to be freed are not allocated to the current task, abnormally terminate the caller.

FREEMAIN LU, LA=LNTHLIST, A=AREAADD, SP=5

. .

LNTHLIST DC F'200', F'800', X'80', FL3'32'

AREAADD DS 3F

**FREEMAIN: List form**

Use the list form of the FREEMAIN macro to construct a nonexecutable control program parameter list.

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

```assembly
name name: symbol. Begin name in column 1.
```

b One or more blanks must precede FREEMAIN.
FREEMAIN macro

FREEMAIN

One or more blanks must follow FREEMAIN.

<table>
<thead>
<tr>
<th>LC</th>
<th>LU</th>
<th>L</th>
<th>VC</th>
<th>VU</th>
<th>V</th>
<th>EC</th>
<th>EU</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>,LA=</td>
<td>length addr</td>
<td>length addr: A-type address.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>,LV=</td>
<td>length value</td>
<td>length value: symbol or decimal number.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Notes:**
1. LA may only be specified with LC, LU, or L above.
2. LV may only be specified with EC, EU, or E above.

| .A=|addr|addr: A-type address.|
| .SP=|subpool nmbr|subpool nmbr: symbol or decimal number.|
| .RELATED=|value|value: any valid assembler character string.|
| .MF=L|

**Parameters**

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

| .MF=L|

Specifies the list form of the FREEMAIN macro.

**FREEMAIN: Execute form**

A remote control program parameter list is used in, and can be modified by, the execute form of the FREEMAIN macro. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN.

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

| name| name: symbol. Begin name in column 1.|
FREEMAIN macro

b One or more blanks must precede FREEMAIN.

FREEMAIN

b One or more blanks must follow FREEMAIN.

LC
LU
L
VC
VU
V
EC
EU
E

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

Notes:
1. LA may only be specified with LC, LU, or L above.
2. LV may only be specified with EC, EU, or E above.

,A=addr  addr: RX-type address, or register (2) - (12).
,SP=subpool nmbr  subpool nmbr: symbol, decimal number, or register (0) or (2) - (12).
,RELATED=value  value: any valid assembler character string.
,MF=(E,list addr)  list addr: RX-type address, or register (1) or (2) - (12).

Parameters

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

,MF=(E,list addr)

Specifies the execute form of the FREEMAIN macro using a remote control program parameter list.
Chapter 98. GETMAIN — Allocate virtual storage

Description

Use the GETMAIN macro to request one or more areas of virtual storage.

Before obtaining storage, be sure to read the information about subpools in the virtual storage management chapter in z/OS MVS Programming: Assembler Services Guide.

You can also use the STORAGE macro to obtain storage. Compared to GETMAIN, STORAGE provides an easier-to-use interface and has fewer restrictions. If your program is running in AR-mode or cross-memory mode, use the STORAGE macro to obtain storage.

Notes:

1. When you obtain storage, the system clears the requested storage to zeros if you obtain either:
   - 8192 bytes or more from a pageable, private storage subpool.
   - 4096 bytes or more from a pageable, private storage subpool, with BNDRY=PAGE specified.

   In all other cases you must not assume that the storage is cleared to zeros. The caller can specify CHECKZERO=YES to detect these and other cases where the system clears the requested storage to zeros.

If you use GETMAIN to request real storage backing above 2 gigabytes, but your system does not support 64-bit storage, your request will be treated as a request for backing above 16 megabytes, even on earlier releases of z/OS that do not support backing above 2 gigabytes. However, boundary requirements indicated by the CONTBDY and STARTBDY parameters will be ignored by earlier releases of z/OS.

Environment

The requirements for the caller are:


For subpools 131 and 132: a PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be obtained.

Dispatchable unit mode: Task.

Cross memory mode: PASN=HASN=SASN.

AMODE: 24- or 31-bit.

- For R, LC, LU, VC, VU, EC, or EU requests: If the calling program is in 31-bit mode, the system treats all addresses and values as 31-bit. Otherwise, the system treats addresses and values as 24-bit.
- For RC, RU, VRC, and VRU requests: The system treats all addresses and values as 31-bit.

ASC mode: Primary.

Interrupt status: Enabled for I/O and external interrupts.

Locks: No locks held.
GETMAIN macro

Control parameters: For LC, LU, VC, VU, EC, EU requests: control parameters must be in the primary address space.

For other requests: control parameters are in registers.

Programming requirements
None.

Restrictions
- For SVC entry, the caller cannot have an EUT FRR established.

Input register information
Before issuing the GETMAIN 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 LC, LU, VC, VU, EC, and EU requests: when control returns to the caller, the general purpose registers (GPRs) contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system.</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged.</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>15</td>
<td>Contains the return code.</td>
</tr>
</tbody>
</table>

For RC, RU, and R requests: when control returns to the caller the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.</td>
</tr>
</tbody>
</table>

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

| 2-13     | Unchanged. |
| 14       | Used as a work register by the system. |
| 15       | Contains the return code. |

For VRC and VRU requests: when control returns to the caller the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system.</td>
</tr>
<tr>
<td>1</td>
<td>The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.</td>
</tr>
</tbody>
</table>

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

| 2-13     | Unchanged. |
| 14       | Used as a work register by the system. |
| 15       | Contains the return code. |

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

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

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

Performance implications
Repeatedly issuing the GETMAIN macro can slow down performance. If your program requires many identically sized storage areas, use the CPOOL macro or callable cell pool services for better performance.

Syntax
The standard form of the GETMAIN macro is written as follows:

```
name

b
GETMAIN

b
```

```
LC,LA=length addr,A=addr
LU,LA=length addr,A=addr
VC,LA=length addr,A=addr
VU,LA=length addr,A=addr
EC,LV=length value,A=addr
EU,LV=length value,A=addr
RC,LV=length value
RU,LV=length value
R,LV=length value
VRC,LV=(maximum length value, minimum length value)
VRU,LV=(maximum length value, minimum length value)
,SP=subpool nmbr
```

length addr: A-type address, or register (2) - (12).
length value: symbol, decimal number, or register (2) - (12).
addr: A-type address or register (2) - (12).

Note: RC, RU, VRC, or VRU must be used for address greater than 16 megabytes.

subpool nmbr: symbol or decimal number 0-127, 131, 132; or register (2) - (12).
GETMAIN macro

Default: SP=0
Note: Specify the subpool as follows:
- Use the SP parameter for LC, LU, VC, VU, EC, EU, RC, RU, VRC, and VRU requests, and for R requests where LV does not indicate register 0.
- Use register 0 for R requests with LV=(0); do not code the SP parameter.
The low-order three bytes of register 0 must contain the length of the requested storage, and the high-order byte must contain the subpool number.

Default: BNDRY=DBLWD
Note: BNDRY=DBLWD

Default: BNDRY=PAGE
Note: This parameter may not be specified with R above.

containing_bdy: Decimal number 3-31 or register (2) - (12).
Note: CONTBDY may be specified only with RC or RU.

starting_bdy: Decimal number 3-31 or register (2) - (12).
Note: STARTBDY may be specified only with RC or RU.

key number: decimal numbers 0-15, or register (2) - (12).
Note: KEY may be specified only with RC, RU, VRC, or VRU.

Default: LOC=RES
Note: You must specify the INADDR parameter with EXPLICIT.

stor addr: RX-type address or register (1)-(12).
Note: This parameter can only be specified with LOC=EXPLICIT.

Default: CHECKZERO=NO
Note: CHECKZERO may be specified only with RC, RU, VRC, or VRU.

value: Any valid assembler character string

Parameters
The parameters are explained as follows.

The first parameter of the GETMAIN macro is positional and is required. This parameter describes the type or mode of the GETMAIN request. The first parameter can be one of the following values:

LC, LA=length addr, A=addr
GETMAIN macro

LU, LA=length addr, A=addr
VC, LA=length addr, A=addr
VU, LA=length addr, A=addr
EC, LV=length value, A=addr
EU, LV=length value, A=addr
RC, LV=length value
RU, LV=length value
R, LV=length value
VRC, LV=(maximum length value, minimum length value)
VRU, LV=(maximum length value, minimum length value)

LC and LU indicate conditional (LC) and unconditional (LU) list requests, and specify requests for one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas is returned in a list beginning at the address specified in the A parameter. No virtual storage is allocated unless all of the requests in the list can be satisfied.

VC and VU indicate conditional (VC) and unconditional (VU) variable requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is between the two values at the address specified in the LA parameter. The address and actual length of the allocated virtual storage area are returned by the system at the address indicated in the A parameter.

EC and EU indicate conditional (EC) and unconditional (EU) element requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is indicated by the parameter, LV=length value. The address of the allocated virtual storage area is returned at the address indicated in the A parameter.

RU and R indicate unconditional register requests; RC indicates a conditional register request. RC, RU, and R specify requests for single areas of virtual storage. The length of the single virtual area is indicated by the parameter, LV=length value. The address of the allocated virtual storage area is returned in register 1.

VRC and VRU indicate variable register conditional (VRC) and unconditional (VRU) requests for a single area of virtual storage. The length returned will be between the maximum and minimum lengths specified by the parameter LV=(maximum length value, minimum length value). The address of the allocated virtual storage is returned in register 1 and the length in register 0.

Notes:
1. A **conditional request** indicates that the active unit of work is not to be abnormally terminated if there is insufficient contiguous virtual storage to satisfy the request. A conditional request does not prevent all abnormal terminations. For example, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. An **unconditional request** indicates that the active unit of work is to be abnormally terminated whenever the request cannot complete successfully.
2. The LC, LU, VC, VU, EC, EU, and R requests can be used only to obtain virtual storage with addresses below 16 megabytes. The RC, RU, VRC, and VRU requests can be used to obtain virtual storage with addresses above 16 megabytes.

LA specifies the virtual storage address of consecutive fullwords starting on a fullword boundary. Each fullword must contain the required length in the low-order three bytes, with the high-order byte set to 0. The lengths should be multiples of 8; if they are not, the system uses the next higher multiple of 8. If
VC or VU was coded, two words are required. The first word contains the
minimum length required, the second word contains the maximum length. If LC
or LU was coded, one word is required for each virtual storage area requested;
the high-order bit of the last word must be set to 1 to indicate the end of the list.
The list must not overlap the virtual storage area specified in the A parameter.

LV=\textit{length value} specifies the length, in bytes, of the requested virtual storage.
The number should be a multiple of 8; if it is not, the system uses the next
higher multiple of 8. If R is specified, LV=(0) may be coded; the low-order three
bytes of register 0 must contain the length, and the high-order byte must
contain the subpool number. LV=(\textit{maximum length value}, \textit{minimum length
value}) specifies the maximum and minimum values of the length of the storage
request.

The \textit{A} parameter specifies the virtual storage address of consecutive fullwords,
starting on a fullword boundary. The system places the address of the virtual
storage area allocated in one or more words. If E was coded, one word is
required. If LC or LU was coded, one word is required for each entry in the LA
list. If VC or VU was coded, two words are required. The first word contains the
address of the virtual storage area, and the second word contains the length
actually allocated. The list must not overlap the virtual storage area specified in
the LA parameter.

\texttt{,SP=\textit{subpool \textit{nmbr}}}

Specifies the number of the subpool from which the virtual storage area is to be
allocated. If you specify a register, the subpool number must be in bits 24-31 of
the register, with bits 0-23 set to zero. Valid subpool numbers are 0-127, 131,
and 132. See the virtual storage management chapter in \textit{z/OS MVS Programming:
Assembler Services Guide} for complete information about these
subpools.

\texttt{,BNDRY=DBLWD}

\texttt{,BNDRY=PAGE}

Specifies that alignment on a doubleword boundary (DBLWD) or alignment with
the start of a virtual page on a 4K boundary (PAGE) is required for the start of a
requested area.

\texttt{,CONTBDY=\textit{containing \textit{bdy}}}

Specifies the boundary the obtained storage must be contained within. Specify
a power of 2 that represents the containing boundary. Supported values are
3-31. For example, CONTBDY=10 means the containing boundary is \textit{2**10}, or
1024 bytes. The containing boundary must be at least as large as the maximum
requested boundary. The obtained storage will not cross an address that is a
multiple of the requested boundary.

If a register is specified, the value must be in bits 24 - 31 of the register.
CONTBDY is valid only with RC or RU.

CONTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

CONTBDY applies to all subpools.

Generally, if you omit this parameter, there is no containing boundary. However,
if the \texttt{GETMAIN} is for SQA or LSQA, and is for less than 4 KB, and \texttt{STARTBDY}
is specified, the default of CONTBDY is 12, ensuring that the \texttt{GETMAIN} stays
within a 4 KB page boundary.

For \texttt{GETMAIN} macros that specify a CONTBDY parameter value that is larger
than 12, it is possible that the allocated area spans across a 4 KB page
boundary, even when the area is less than or equal to 4 KB and in an SQA or
LSQA subpool.
GETMAIN macro

,STARTBDY=starting_bdy
   Specifies the boundary the obtained storage must start on. Specify a power of 2
   that represents the start boundary. Supported values are 3-31. For example,
   STARTBDY=10 means the start boundary is \(2^{10}\), or 1024 bytes. The obtained
   storage will begin on an address that is a multiple of the requested boundary.

   If a register is specified, the value must be in bits 24-31 of the register.
   STARTBDY is valid only with RC or RU.

   STARTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

   STARTBDY applies to all subpools.

   If you omit this parameter, the start boundary is 8 bytes (equivalent to
   specifying STARTBDY=3).

,KEY=key number
   Specifies the storage key in which the storage is to be obtained. The valid
   storage keys are 0-15. If a register is specified, the storage key must be in bits
   24-27 of the register. KEY is valid with RC, RU, VRC, or VRU, and applies to
   subpools 131 and 132 only. See the virtual storage management chapter in
   z/OS MVS Programming: Assembler Services Guide
   for information about how
   the system assigns the storage key for your storage request.

,LOC=24
 ,LOC=(24,31)
 ,LOC=(24,64)
 ,LOC=31
 ,LOC=(31,31)
 ,LOC=(31,64)
 ,LOC=RES
 ,LOC=(RES,31)
 ,LOC=(RES,64)
 ,LOC=EXPLICIT
 ,LOC=(EXPLICIT,24)
 ,LOC=(EXPLICIT,31)
 ,LOC=(EXPLICIT,64)
   Specifies the location of virtual storage and central (also called real) storage.
   This is especially helpful for callers with 24-bit dependencies. When LOC is
   specified, central storage is allocated anywhere until the storage is fixed. You
   can specify the location of central storage (after the storage is fixed) and virtual
   storage (whether or not the storage is fixed) using the following LOC parameter
   values:

   LOC=24 indicates that central and virtual storage are to be located below 16
   megabytes.

   Notes:
   1. Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW
      is still supported, but IBM recommends using LOC=24 instead.
   2. LOC=24 should not be used to allocate disabled reference (DREF) storage. 
      If issued in AMODE24, an abend B78 will result. In AMODE31, the LOC=24
      parameter will be ignored, and the caller will be given an address above 16
      megabytes.

   LOC=(24,31) indicates that virtual storage is to be located below 16 megabytes
   and central storage can be located anywhere below 2 gigabytes.
**GETMAIN macro**

**Note:** Specifying LOC=(BELOW,ANY) is the same as specifying LOC=(24,31). LOC=(BELOW,ANY) is still supported, but IBM recommends using LOC=(24,31) instead.

LOC=(24,64) indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere in 64-bit storage.

LOC=31 and LOC=(31,31) indicate that virtual and central storage can be located anywhere below 2 gigabytes.

**Note:** Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC=31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64) indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

**Note:** When you specify LOC=31, GETMAIN tries to allocate virtual storage above 16 megabytes. If the attempt fails, GETMAIN tries to allocate virtual storage below 16 megabytes. If this attempt also fails, GETMAIN does not allocate any storage.

When you use LOC=RES to allocate storage that can reside either above or below 16 megabytes, LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be located below 16 megabytes. If the caller resides above 16 megabytes, virtual and central storage are to be located either above or below 16 megabytes.

LOC=(RES,31) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere below 2 gigabytes. In either case, central storage can be located anywhere below 2 gigabytes.

**Note:** Specifying LOC=(RES,ANY) is the same as specifying LOC=(RES,31). LOC=(RES,ANY) is still supported, but IBM recommends using LOC=(RES,31) instead.

LOC=(RES,64) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere in 31-bit storage. In either case, central storage can be located anywhere in 64-bit storage.

LOC=EXPLICIT, LOC=(EXPLICIT,24), LOC=(EXPLICIT,31), or LOC=(EXPLICIT,64) specify that the requested virtual storage is to be located at the address specified with the INADDR parameter, which is required with EXPLICIT. EXPLICIT is valid only for subpools 0-127, 131, and 132. You can use LOC=EXPLICIT only with RC or RU. You cannot specify the BNDRY parameter with EXPLICIT.

**Note:** Specifying LOC=(EXPLICIT,BELOW) is the same as specifying LOC=(EXPLICIT,24). Specifying LOC=(EXPLICIT,ANY) is the same as specifying LOC=(EXPLICIT,31). The older specifications are still supported, but IBM recommends using the newer specifications instead.

LOC=(EXPLICIT,31) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere below 2 gigabytes.
LOC=(EXPLICIT,24) indicates that virtual storage is to be located at the address specified on the INADDR parameter, and central storage is to be located below 16 megabytes. The virtual storage address specified on the INADDR parameter must be below 16 megabytes.

LOC=EXPLICIT and LOC=(EXPLICIT,64) indicate that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere in 64-bit storage.

When you specify EXPLICIT on a request for storage from the same virtual page as previously requested storage, you must request it in the same key, subpool, and central storage area as on the previous storage request. For example, if you request virtual storage backed with central storage below 16 megabytes, any subsequent requests for storage from that virtual page must be specified as LOC=(EXPLICIT,24).

INADDR=stor addr
Specifies the desired virtual address for the storage to be obtained. When you specify INADDR, you must specify EXPLICIT on the LOC parameter.

Notes:
1. The address specified on INADDR must be on a doubleword boundary.
2. Make sure that the virtual storage address specified on INADDR and the central storage backing specified on the LOC=EXPLICIT parameter are a valid combination. For example, if the address specified on INADDR is for virtual storage above 16 megabytes, specify LOC=EXPLICIT or LOC=(EXPLICIT,ANY). Valid combinations include:
   • Virtual above, central any
   • Virtual any, central any
   • Virtual below, central below
   • Virtual below, central any

CHECKZERO=YES
CHECKZERO=NO
Specifies whether or not the return code for a successful completion should indicate if the system has cleared the requested storage to zeroes. When CHECKZERO=NO is specified or defaulted, the return code for a successful completion is 0. When CHECKZERO=YES is specified, the return code for a successful completion is X’14’ if the system has cleared the requested storage to zeroes, and 0 if the system has not cleared the requested storage to zeroes.

There is no performance cost to specifying CHECKZERO=YES.

Programs that issue the GETMAIN macro with the CHECKZERO parameter can run on any z/OS system. On a down-level system, CHECKZERO will be ignored, and the return code for a successful completion (conditional or unconditional) will be 0.

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

ABEND codes
Abend codes the GETMAIN macro might issue are listed below in hexadecimal. For detailed abend code information, see [z/OS MVS System Codes]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>104</td>
<td>10A</td>
<td>178</td>
<td>204</td>
</tr>
</tbody>
</table>
Return and reason codes

When the GETMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| **0**       | **Meaning:** Successful completion. CHECKZERO=YES was not specified, or the system has not cleared the requested storage to zeroes.  
**Action:** None. |
| **4**       | **If you did not specify EXPLICIT on the LOC parameter:**  
**Meaning:** Environmental or system error. Virtual storage was not obtained because insufficient storage is available.  
**Action:** If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit.  
**If you specified EXPLICIT on the LOC parameter:**  
**Meaning:** Program error. Virtual storage was not obtained because part of the requested storage area is outside the bounds of the user region.  
**Action:** Determine why your program is mistakenly requesting storage outside the user region. If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit. |
| **8**       | **Meaning:** System error. Virtual storage was not obtained because the system has insufficient central storage to back the request.  
**Action:** Report the problem to the system programmer so the cause of the problem can be determined and corrected. |
| **C**       | **Meaning:** System error. Virtual storage was not obtained because the system cannot page in the page table associated with the storage to be allocated.  
**Action:** Report the problem to the system programmer so the cause of the problem can be determined and corrected. |
### Table 20. Return Codes for the GETMAIN Macro (continued)

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 10          | **Meaning:** Program error. Virtual storage was not obtained for one of the following reasons: This reason code applies only to GETMAIN requests with LOC=EXPLICIT specified.  
  - Part of the requested area is allocated already.  
  - Virtual storage was already allocated in the same page as this request, but one of the following characteristics of the storage was different:  
    - The subpool  
    - The key  
    - Central storage backing  
  **Action:** Determine why your program is attempting to obtain allocated storage or why your program is attempting to obtain virtual storage with different attributes from the same page of storage. Correct the coding error. |
| 14          | **Meaning:** Successful completion. The system has cleared the requested storage to zeroes. This return code occurs only when CHECKZERO=YES is specified.  
  **Action:** None. |

**Example 1**

Obtain 400 bytes of storage from subpool 10. If the storage is available, the address will be returned in register 1 and register 15 will contain 0; if storage is not available, register 15 will contain 4.

```
GETMAIN RC,LV=400,SP=10
```

**Example 2**

Obtain 48 bytes of storage from default subpool 0. If the storage is available, the address will be stored in the word at AREAADDR; if the storage is not available, the task will be abnormally terminated.

```
GETMAIN EU,LV=48,A=AREAADDR
```

**Example 3**

Obtain a minimum of 1024 bytes to a maximum of 4096 bytes of virtual storage from default subpool 0 with virtual and central storage locations either above or below 16 megabytes. If the storage is available, the starting address is to be returned in register 1 and the length of the storage allocated is to be returned in register 0; if the storage is not available, the caller is to be terminated.

```
GETMAIN VRU,LV=(4096,1024),LOC=ANY
```

### GETMAIN—List form

Use the list form of the GETMAIN macro to construct a control program parameter list. The list form of the GETMAIN macro cannot be used to allocate virtual storage with addresses greater than 16 megabytes.

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

name: Begin name in column 1.

b

One or more blanks must precede GETMAIN.

GETMAIN

One or more blanks must follow GETMAIN.

LC
LU
VC
VU
EC
EU

,LA=length addr
,LV=length value

length addr: A-type address.
length value: symbol or decimal number.

Notes:
1. LA may not be specified with EC or EU above.
2. LV may not be specified with LC, LU, VC or VU above.

,A=addr

addr: A-type address.

,SP=subpool nmbr

subpool nmbr: symbol or decimal number 0-127, 131, 132.

Default: SP=0
Note: Use the SP parameter for LC, LU, VC, VU, EC, and EU requests.

,BNDRY=DBLWD
,BNDRY=PAGE

Default: BNDRY=DBLWD

,RELATED=value

value: any valid assembler character string.

,MF=L

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

,MF=L

Specifies the list form of the GETMAIN macro.
GETMAIN—Execute form

A remote control program parameter list is used in, and can be modified by, the execute form of the GETMAIN macro. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN. The execute form of the GETMAIN macro cannot be used to allocate virtual storage with addresses greater than 16 megabytes.

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

```
name          name: symbol. Begin name in column 1.
b             One or more blanks must precede GETMAIN.
GETMAIN
b             One or more blanks must follow GETMAIN.
LC, LU, VC, VU, EC, EU
,LA= length addr  length addr: RX-type address or register (2) - (12).
,LV= length value  length value: symbol, decimal number, or register (2) - (12).
Notes:
1. LA may not be specified with EC or EU above.
2. LV may not be specified with LC, LU, VC, or VU above.
,A= addr        addr: RX-type address, or register (2) - (12).
,SP= subpool nmbr  subpool nmbr: symbol; decimal number 0-127, 131, 132; or register (2) - (12).
Default: SP=0
Note: Specify the subpool as follows:
• Use the SP parameter for LC, LU, VC, VU, EC, EU, RC, RU, VRC, and VRU requests, and for R requests where LV does not indicate register 0.
• Use register 0 for R requests with LV=(0); do not code the SP parameter. The low-order three bytes of register 0 must contain the length of the requested storage, and the high-order byte must contain the subpool number.
,BNDRY=DBLWD     Default: BNDRY=DBLWD
,BNDRY=PAGE
```
GETMAIN macro

\[ \text{,RELATED=} \text{value} \quad \text{value}: \text{any valid assembler character string.} \]

\[ \text{,MF=} \{ \text{E, list addr} \} \quad \text{ctrl prog}: \text{RX-type address, or register (1) or (2) - (12).} \]

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

\[ \text{,MF=} \{ \text{E, list addr} \} \quad \text{Specifies the execute form of the GETMAIN macro using a remote control program parameter list.} \]
Chapter 99. GQSCAN — Extract information from global resource serialization queue

Description

Use the GQSCAN macro to obtain the status of resources and requestors of those resources. The GQSCAN macro allows you to obtain resource information from the system.

ISGQUERY is the IBM recommended replacement for the GQSCAN service.

The ISGRIIB macro allows you to interpret the data that the GQSCAN service routine returns to the user-specified area. The ISGRIIB macro maps the resource information block (RIB) and the resource information block extent (RIBE) as shown in z/OS MVS Data Areas in z/OS Internet Library at [http://www.ibm.com/systems/z/os/zos/bkserv/](http://www.ibm.com/systems/z/os/zos/bkserv).

There are two fields in the RIB that you can use to determine whether any RIBEs were not returned:

- RIBTRIBE contains the total number of RIBEs associated with this RIB
- RIBNRIIB contains the total number of RIBEs returned by GQSCAN with this RIB in the user-specified area indicated by the AREA parameter.

Global resource serialization counts and limits the number of outstanding global resource serialization requests. A global resource serialization request is any ENQ, RESERVE, or GQSCAN that causes an element to be inserted into a queue in the global resource serialization request queue area.

Environment

The requirements for the caller are:

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

Programming requirements

To interpret the data that the GQSCAN service routine returns in the user-specified area, you must include the ISGRIIB mapping macro as a DSECT in your program.

Restrictions

Unauthorized callers of GQSCAN need to be authorized through Security Authorization Facility (SAF) when Multi-level security (MLS) is active. If the caller is not authorized, the request will fail.
When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=QSCAN must have at least READ authorization to the ISG.QSCANSERVICES.AUTHORIZATION resource in the FACILITY class. You can activate the multilevel security support through the SETROPTS MLACTIVE option in RACF. For general information about defining profiles in the FACILITY class, see z/OS Security Server RACF Command Language Reference and z/OS Security Server RACF Security Administrator's Guide. For information about multilevel security, see z/OS Planning for Multilevel Security and the Common Criteria.

Input register information
Before issuing the GQSCAN macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information
When control returns to the caller, the GPRs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Register 0 contains a fullword reason code if the return code in register 15 is X'0A' or X'0C'. Otherwise, register 0 contains the following two halfword values:</td>
</tr>
<tr>
<td></td>
<td>• The first (high-order) halfword contains the length of the fixed portion of each RIB returned.</td>
</tr>
<tr>
<td></td>
<td>• The second (low-order) halfword contains the length of each RIBE returned or reason code.</td>
</tr>
<tr>
<td>1</td>
<td>Contains the number of RIBs that were copied into the area provided</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

When control returns to the caller, the ARs contain:

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
In general, the narrower the search parameters (particularly QNAME and RNAME), the less time it takes. Using both a specific QNAME and a specific RNAME gives better performance than using generic prefix.

The use of XSYS=YES (the default) might greatly degrade the performance of the request. Use it only when required.
Polling for ENQ contention through GQSCAN or ISGQUERY is not recommended. See the [z/OS MVS Planning: Global Resource Serialization](https://www.ibm.com) and [z/OS MVS Programming: Authorized Assembler Services Guide](https://www.ibm.com) for more information about monitoring contention through ENF 51.

None.

**Syntax**

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

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

b
One or more blanks must precede GQSCAN.

GQSCAN

b
One or more blanks must follow GQSCAN.

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

,REQLIM=value
value: symbol, decimal digit, register (2) - (12), or the word MAX.
Default: REQLIM=MAX

,REQLIM=MAX

,SCOPE=ALL
Default: SCOPE=STEP

,SCOPE=STEP

,SCOPE=SYSTEM

,SCOPE=SYSTEMS

,RESERVE=YES
Default: All resources requested with RESERVE and all resources requested with ENQ.

,RESERVE=NO

,RESERVE=NO

,RESNAME=(qname addr,mname addr,
mname length],
[GENERIC|SPECIFIC],
qname length)
qname addr: RX-type address or register (2) - (12).
mname addr: RX-type address or register (2) - (12).
mname length: decimal digit, or register (2) - (12).
Default: assembled length of mname.
Default: qname length of eight.

,SYSTYPE=(sysname addr
[.asid value])
sysname addr: RX-type address or register (2) - (12).
asid value: symbol, decimal digit, or register (2) - (12).
Note: Provide mname addr only when qname addr is used. Code mname length if a register is specified for mname addr. Code an asid value only when the sysname addr is used.

,QUIT=YES
Default: QUIT=NO

,QUIT=NO
Note: QUIT=YES is mutually exclusive with all parameters but TOKEN and MF.

,REQCNT=value
value: decimal digit or register (2) - (12).
```
**Parameters**

The parameters are explained as follows:

- **AREA=(area addr,area size)**
  Specifies the location and size of the area where information extracted from the global resource serialization resource queues is to be placed. The minimum size is the amount needed to describe a single resource, which is the length of the fixed portions of the RIB and the maximum size rounded up to a fullword value. IBM recommends that you use a minimum of 1024 bytes as the area size.

- **REQLIM=value**
  **REQLIM=MAX**
  Specifies the maximum number of owners and waiters to be returned for each individual resource within the specification of RESNAME, which can be any value between 0 and $2^{15}-1$. MAX specifies $2^{15}-1$ (32767).

- **SCOPE=ALL**
  **SCOPE=STEP**
  **SCOPE=SYSTEM**
  **SCOPE=SYSTEMS**
  Specifies that you want information only for resources having the indicated scope. STEP, SYSTEM, or SYSTEMS is the scope specified on the resource request. If you specify SCOPE=ALL (meaning STEP, SYSTEM, and SYSTEMS), the system returns information for all resources the system recognizes that have the specified RESNAME, RESERVE, or SYSNAME characteristics.

- **RESERVE=YES**
  **RESERVE=NO**
  If you specify RESERVE=YES, information is only returned for the requestors of the resource, that requested the resource with the RESERVE macro. If, for example, the resource also had requestors with the ENQ macro, the ENQ requestor’s information would not be returned for the resource.

RESERVE=NO information is only returned for the requestors of the resource that requested the resource with the ENQ macro. In other words, if the resource...
also had requestors with the RESERVE macro, the RESERVE requestor's information would not be returned for the resource.

```
RESNAME=(qname addr[,rname addr,rname length][,GENERIC|SPECIFIC], qname length)
```

RESNAME identifies an individual resource or group of resources that GQSCAN will examine.

RESNAME with (rname) indicates the name of one resource. The qname addr specifies the address of the 8-character major name of the requested resource.

The rname addr specifies the virtual storage address of a 1 to 255-byte minor name used with the major name to represent a single resource. Information returned is for a single resource unless you specify SCOPE=ALL, in which case it could be for three resources (STEP, SYSTEM, and SYSTEMS). If the name specified by rname is defined by an EQU assembler instruction, the rname length must be specified.

The rname length specifies the length of the minor name. If you use the register form, specify length in the low-order (rightmost) byte. The length must match the rname length specified on ENQ or RESERVE.

GENERIC specifies that the rname of the requested resource must match but only for the length specified. For example, an ENQ for SYS1.PROCLIB would match the GQSCAN name specified as SYS1 for an rname length of 4.

SPECIFIC specifies that the name of the requested resource must exactly match the GQSCAN name.

**Note:** GENERIC and SPECIFIC are mutually exclusive.

The qname length specifies the number of characters in a resource qname that must match the GQSCAN qname specified by RESNAME. You must specify a qname length to request a GQSCAN for a generic qname. For example, an ENQ with a qname of SYSDSN would match a GQSCAN specifying GENERIC with a qname of SYSD and qname length of 4. Specify zero for the qname length (with any qname) to request a generic GQSCAN matching any resource qname. If you do not specify a qname length, GQSCAN uses the default of 8.

```
SYSNAME=(sysname addr [,asid value])
```

Specify SYSNAME to tell GQSCAN to return information for resources requested by tasks running on the MVS system specified in an 8-byte field pointed to by the address in sysname address and the asid value, a 4-byte address space identifier, right justified. Valid SYSNAMEs are specified in the IEASYSxx parmlib member.

Information returned includes only those resources whose sysname addr and asid value match the ones specified. SYSNAME=0 or SYSNAME=(0,asid value), specifies that the system name is that of the system on which GQSCAN is issued. The system issues return code X'0A' with a reason code of X'0C', if SYSNAME≠0 or SYSNAME=(0,asid value) is specified with XSYS=NO.

```
QUIT=YES
QUIT=NO
```

QUIT=NO indicates that you do not want to end the current global resource serialization queue scan. QUIT=YES tells GQSCAN to stop processing the current global resource serialization queue scan and release the storage allocated to accumulate the information specified in the token.
GQSCAN macro

If you specify QUIT=YES, you must specify the TOKEN parameter. If you specify QUIT=YES without the TOKEN parameter, the system issues abend X'09A'.

,REQCNT=rcount
Specifies that you want GQSCAN to return resource information only when the total number of requesters (owners plus waiters) for an individual resource is greater than or equal to rcount, which can be any value between 0 and 2^{31}-1.

,OWNERCT=ocount
Specifies that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to ocount, which can be any value between 0 and 2^{31}-1.

,WAITCNT=wcount
Specifies that you want GQSCAN to return resource information only when the total number of waiters for an individual resource is greater than or equal to wcount, which can be any value between 0 and 2^{31}-1.

,OWNERCT=ocount,WAITCNT=wcount
Specifies that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to ocount or when the total number of waiters for an individual resource is greater than or equal to wcount.

,TOKEN=addr
Specifies the address of a fullword of storage that the GQSCAN service routine can use to provide you with any remaining information in subsequent invocations. If the token value is zero, the scan starts at the beginning of the resource queue. If the token value is not zero, the scan resumes at the point specified on TOKEN. Specify the same token value that GQSCAN returned on its previous invocation to continue where processing left off on the previous invocation.

When providing a non-zero token value, you must specify the same scope that you specified on the GQSCAN request that returned the token.

,XSYS=YES
,XSYS=NO
Specifies whether GQSCAN should be propagated across systems in the global resource serialization complex, to gather complex-wide information. This parameter is ignored in a global resource serialization ring complex, and for requests that only gather local data.

Specify XSYS=YES if the program requires complex-wide global resource serialization information. The caller might be suspended while the information is being gathered. Do not specify or default to XSYS=YES if this condition cannot be tolerated.

Specify XSYS=NO if the program will accept global resource serialization information from this system only. The RIBE data will contain information about requestors from other other systems in the complex only if that information is already available on the GQSCAN caller's system. Otherwise, RIBE data will be provided only for requests from the GQSCAN caller's system, and the counts in the RIB will reflect only those requests. This request is always handled without placing the caller's dispatchable unit into a wait.

ABEND codes
See [z/OS MVS System Codes](https://www.ibm.com) for more information about the abend codes.
Return and reason codes

When GQSCAN returns control, register 15 contains one of the following return codes:

Table 21. Return Codes for the GQSCAN Macro

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td><strong>Meaning</strong>: Queue scan processing is complete. Data is now in the area you specified. There is no more data to return. <strong>Action</strong>: Process the data.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Meaning</strong>: Queue scan processing is complete. No resources matched your request. <strong>Action</strong>: Do not try to process any data; none exists.</td>
</tr>
<tr>
<td>8</td>
<td><strong>Meaning</strong>: The area you specified was filled before queue scan processing completed. <strong>Action</strong>: If you specified TOKEN, process the information in the area and issue GQSCAN again, specifying the TOKEN returned to you. If you did not specify TOKEN, specify a larger area or specify a TOKEN.</td>
</tr>
</tbody>
</table>
Table 21. Return Codes for the GQSCAN Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0A</td>
<td>Meaning: The information you specified to GQSCAN is not valid. Action: Take the action indicated by the following hexadecimal reason code found in register 0.</td>
</tr>
<tr>
<td>04</td>
<td>The caller attempted to use GQSCAN before the global resource serialization (GRS) address space was active.</td>
</tr>
<tr>
<td>08</td>
<td>The size of the reply area, specified by the AREA parameter, is too small to contain a resource information block (RIB) of maximum size.</td>
</tr>
<tr>
<td>0C</td>
<td>You specified mutually exclusive arguments (RESERVE=YES, RESERVE=NO, RESNAME=, SYSNAME=, or XSYS=NO) to GQSCAN.</td>
</tr>
<tr>
<td>10</td>
<td>The caller was holding a local lock other than the GRS local lock when GQSCAN was invoked.</td>
</tr>
</tbody>
</table>
| 14                      | One of the following conditions, in reference to the RESNAME parameter, was detected by GQSCAN:  
  - The qname length was specified with a value greater than eight.  
  - The qname length value was specified without the qname addr value.  
  - The SPECIFIC parameter was specified with a name length value of zero.  
  - The name or name length was specified without the qname addr value. |
| 18                      | The asid value, for the SYSNAME parameter was specified without the sysname addr value. |
| 1C                      | The REQCNT parameter was specified with either the OWNERCNT or WAITCNT parameters. |
| 20                      | The combination of values specified on the SCOPE parameter is not valid. |
| 28                      | An element in GQSCAN's input parameter list was not in the caller's storage protect key. |
| 2C                      | An invalid token was specified to GQSCAN. |
| 30                      | The GQSCAN caller is not authorized to use the restricted interface (SCOPE=LOCAL or GLOBAL). |
| 34                      | QUIT=YES was specified without the TOKEN parameter. |
| 38                      | The caller held a CMS lock other than CMSEQDQ when GQSCAN was invoked. |
| 3C                      | The caller held a lock that violated the environmental restrictions of a service required by GQSCAN. |
| 40                      | The caller invoked GQSCAN in the service request block (SRB) mode. |
| 44                      | The value specified for the REQLIM parameter was not valid. |
| 48                      | The value specified for the REQCNT parameter was not valid. |
| 4C                      | The value specified for the OWNERCT parameter was not valid. |
| 50                      | The value specified for the WAITCNT parameter was not valid. |
| 58                      | SETROPTS MLACTIVE is in effect, and the program is not authorized to issue GQSCAN. Ensure the program is running authorized, or is associated with a userid with at least READ access to the best fit FACILITY class resource profile of the form ISG.GSCANSERVICES.AUTHORIZATION and that the FACILITY class is SETROPTS RACLSTed. |
### Table 21. Return Codes for the GQSCAN Macro (continued)

<table>
<thead>
<tr>
<th>Hexadecimal Return Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>System error. Queue scan encountered an abnormal situation while processing. The information in your area is not meaningful. The reason code in register 0 contains one of the following:</td>
</tr>
<tr>
<td></td>
<td>Reason Code</td>
</tr>
<tr>
<td></td>
<td>00</td>
</tr>
<tr>
<td></td>
<td>04</td>
</tr>
<tr>
<td></td>
<td>08</td>
</tr>
<tr>
<td></td>
<td>0C</td>
</tr>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>14</td>
</tr>
<tr>
<td></td>
<td>Action</td>
</tr>
<tr>
<td>10</td>
<td>Program error. An incorrect SYSNAME was specified as input to queue scan. The information in your area is not meaningful.</td>
</tr>
<tr>
<td></td>
<td>Action</td>
</tr>
<tr>
<td>14</td>
<td>Environmental error. The area you specified was filled before queue scan processing completed. Your request specified TOKEN, but the limit for the number of concurrent resource requests (ENQ, RESERVE, or GQSCAN) has been reached. The information in your area is valid but incomplete. The scan cannot be resumed.</td>
</tr>
<tr>
<td></td>
<td>Action</td>
</tr>
</tbody>
</table>

### GQSCAN: List form

The list form of the GQSCAN macro is used to construct a non-executable parameter list. This parameter list, or a copy of it for reentrant programs, can be referred to by the execute form of the GQSCAN macro.

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

```
name name: symbol. Begin name in column 1.
b One or more blanks must precede GQSCAN.
GQSCAN
b One or more blanks must follow GQSCAN.
AREA=(area addr, area size) area addr: A-type address.
area size: symbol, decimal digit.
```
GQSCAN macro

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

,REQLIM=value
,REQLIM=MAX
value: symbol, decimal digit or the word MAX.
Default: REQLIM=MAX

,SCOPE=ALL
,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
Default: SCOPE=STEP

,RESERVE=YES
,RESERVE=NO
Default: All resources requested with RESERVE and all resources requested with ENQ.

,RESNAME=(qname addr [,name addr, mame length], [GENERIC|SPECIFIC], qname length)
qname addr: A-type address.
mame addr: A-type address.
mame length: decimal digit.
Default: assembled length of mame.
Default: qname length of eight.

,SYSNAME=(sysname addr [,asid value])
sysname addr: A-type address.
asid value: symbol, decimal digit.
Note: mame addr can be provided only when qname addr is used. mame length must be provided if a register is specified for mame addr. An asid value can be coded only when the sysname addr is used.

,QUIT=YES
,QUIT=NO
Default: QUIT=NO
Note: Only TOKEN and MF=L can be specified with QUIT=YES.

,REQCNT=value
value: decimal digit.
Default: REQCNT=0

,OWNERCT=value,WAITCNT=value
value: decimal digit.

,OWNERCT=value
value: decimal digit.

,WAITCNT=value
value: decimal digit.

,TOKEN=addr
addr: RX-type address.

,XSYS=YES
,XSYS=NO
Default: XSYS=YES
Note: XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.

,MF=L
Parameters

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

\[MF=L\]

Specifies the list form of the GQSCAN macro.

GQSCAN: Execute form

The execute form of the GQSCAN macro can refer to and modify a remote parameter list built by the list form of the macro. There are no defaults for any of the parameters in the execute form of the macro.

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

```
name name
\:egin{name}
\:Begin\: \name\: in\: column\: 1.

b
One\: or\: more\: blanks\: must\: precede\: GQSCAN.

GQSCAN
b
One\: or\: more\: blanks\: must\: follow\: GQSCAN.
```

\[AREA=(area\: addr,\: area\: size)\]

area addr: RX-type address or register (2) - (12).
area size: symbol, decimal digit, or register (2) - (12).

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

\[,REQLIM=value\]

value: symbol, decimal digit, register (2) - (12), or the word MAX.

\[,SCOPE=STEP\]

Note: SCOPE=LOCAL and SCOPE=GLOBAL cannot be coded on the list form of this macro.

\[,SCOPE=ALL\]

\[,SCOPE=SYSTEM\]

\[,SCOPE=SYSTEMS\]

\[,RESERVE=YES\]

\[,RESERVE=NO\]

\[,RESNAME=(qname\: addr,[name\: addr,\: name\: length]),\]

qname addr: RX-type address or register (2) - (12).
name addr: RX-type address or register (2) - (12).
name length: decimal digit, register (2) - (12). Default: assembled length of name.

\[,SYSTYPE=(sysname\: addr[,\: asid\: value] \)

sysname addr: RX-type address or register (2) - (12).
asid value: symbol, decimal digit, or register (2) - (12).
**GQSCAN macro**

**Note:** `rname addr` can be provided only when `qname addr` is used. `rname length` must be provided if a register is specified for `rname addr`. An `asid value` can be coded only when the `sysname addr` is used.

**,QUIT=YES**  
**Default:** QUIT=NO

**,QUIT=NO**  
**Note:** Only `TOKEN` and `MF=(E, parm list addr)` can be specified with `QUIT=YES`.

**,REQCNT=value**  
`value`: decimal digit or register (2) - (12).  
**Default:** REQCNT=0

**,OWNERCT=value,WAITCNT=value**  
`value`: decimal digit.

**,OWNERCT=value**  
`value`: decimal digit.

**,WAITCNT=value**  
`value`: decimal digit.

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

**,XSYS=YES**  
**Default:** XSYS=YES

**,XSYS=NO**  
**Note:** XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.

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

---

**Parameters**

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

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

`list addr` specifies the area that the system uses to contain the parameters.
Chapter 100. HSPSERV — Read from and write to a hiperspace

Description

HSPSERV transfers data between virtual storage areas in address spaces and hiperspaces. It reads data from a hiperspace to an address space and it writes data to a hiperspace from an address space.

A hiperspace can be either a standard hiperspace, of which there are two types, shared and nonshared, or an ESO (expanded storage only) hiperspace. The nonshared standard hiperspace and the shared standard hiperspace are backed with real storage, and if necessary, with auxiliary storage. Through the buffer area in the address space, your program can view or scroll through the hiperspace. HSPSERV SWRITE and HSPSERV SREAD transfer data to and from a standard hiperspace. For more information about hiperspaces, see z/OS MVS Programming: Assembler Services Guide.

The STOKEN parameter identifies the specific hiperspace to be read from or written to. The HSPALET parameter specifies an optional ALET for the hiperspace. The RANGLIST parameter identifies the storage range in the address space and the storage range in the hiperspace. A storage range consists of contiguous 4K byte blocks starting on a 4K byte boundary.

Environment

The requirements for the caller are:

- **Minimum authorization**: Problem state and any PSW key
- **Dispatchable unit mode**: Task or SRB
- **Cross memory mode**: Any PASN, any HASN, any SASN
  - **Note**: PASN=HASN=SASN is required for a nonshared standard hiperspace for which an ALET is not used (the HSPALET parameter is omitted).
- **AMODE**: 31-bit
- **ASC mode**: Primary or access register (AR)
- **Interrupt status**: Enabled for I/O and external interrupts
- **Locks**: No locks held
- **Control parameters**: Must be in the caller's primary address space. If the caller's PSW key is not zero, the PSW key must match the storage key associated with the control parameters.

Programming requirements

- If you code the HSPALET parameter on the HSPSERV macro, you must first issue the SYSSTATE macro to indicate the ASC mode of your program.
- If you code the HSPALET parameter on the HSPSERV macro, you must provide a 144-byte save area in the caller's primary address space.
- The range list must be addressable in the caller's primary address space.

Restrictions

None.
HSPSERV macro

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

However, if the caller specifies the HSPALET parameter:
- General purpose register (GPR) 13 must contain the address of a 144-byte save area. The save area must be in the caller's primary address space.
- Access register (AR) 13 must contain 0, regardless of whether the caller is in primary or AR address space control (ASC) mode.

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reason code</td>
</tr>
<tr>
<td>1</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14</td>
<td>Used as a work register by the system</td>
</tr>
<tr>
<td>15</td>
<td>Return code</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Register</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>Used as work registers by the system</td>
</tr>
<tr>
<td>2-13</td>
<td>Unchanged</td>
</tr>
<tr>
<td>14-15</td>
<td>Used as work registers by the system</td>
</tr>
</tbody>
</table>

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

Performance implications
None.

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

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

```
name
```

name: Symbol. Begin name in column 1.
### Parameters

The parameters are explained as follows:

**SREAD**

Requests that the system read data from a standard hiperspace to an address space.

STOKEN and RANGLIST are required parameters on the SREAD request. NUMRANGE, RELEASE, HSPALET, RSNCODE, and RETCODE are optional parameters.

**SWRITE**

Requests that the system write data to a standard hiperspace from an address space.

**Notes:**

- When HPSERV returns to the caller after the SWRITE operation, the contents of the address space storage range are not preserved. You can use the address space area again.
- If the hiperspace maps a data-in-virtual object, do not issue an SWRITE request while a DIV SAVE request is current.
STOKEN and RANGLIST are required parameters on the SWRITE request. NUMRANGE, HSPALET, RETCODE, and RSNCODE are optional parameters.

\STOKEN=stoken-addr
Specifies the address of the eight-character variable that contains the STOKEN for the standard hiperspace from which the data is to be read or into which the data is to be written. Restrictions on standard hiperspaces are described in Figure 11 on page 621.

\HSPALET=alet-addr
Specifies either the address of a fullword or a register that contains the ALET for the hiperspace that is to be accessed. The ALET must be for a hiperspace that is on the caller's DU-AL or PASN-AL.

The HSPALET parameter is optional except for the following case: If the calling program accesses a shared hiperspace, is in problem state, and uses PSW key 8 - F, HSPALET is required.

If you code HSPALET, do not code RELEASE=YES.
If you code HSPALET, your recovery routine cannot attempt retry at the time of error.

\NUMRANGE=n
\NUMRANGE=num-addr
Specifies the number of entries, from 1 to 50, or specifies a fullword that identifies the number of entries in the range list (that the RANGLIST parameter points to), or specifies a register containing the address of a fullword containing the number of entries. The default is NUMRANGE=1.

If you omit NUMRANGE, HSPSERV reads or writes one entry in the range list.

\RANGLIST=list-addr
Specifies a fullword that contains an address of a list of ranges that the system is to read or write, or specifies a register that contains the address of the fullword pointer to the range list. The range list consists of a number of entries (specified by NUMRANGE) where each entry specifies (1) a storage location in an address space, (2) a storage location in a hiperspace, and (3) the number of blocks of data the system is to read or write.

Each entry in the range list consists of three words as follows:

First Word  The starting virtual address in the address space into which the data is to be read or from which the data is to be written

Second Word  The starting virtual address in the hiperspace from which the system is to read or into which the system is to write

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

Note that the address is the block number followed by 12 binary zeros.

An example of how to code the RANGLIST parameter when NUMRANGE=3 is as follows:
Restrictions on the areas in the address space and the hiperspace are described in Figure 11 on page 621.

On return, only if the caller issued the HSPSERV macro with the HSPALET parameter, the range list values might be different from the input values if the system could not at first successfully complete the read or write operation. In that case, the system changes the range list values, but does not restore the input values when it finally returns control to the caller.

,**RELEASE=NO**

**RELEASE=YES**

Specifies whether or not the system is to release the hiperspace pages after it completes the SREAD operation. RELEASE is valid only with SREAD.

RELEASE=NO specifies that the system does not release the hiperspace pages after it completes the SREAD operation. (Unless a subsequent SWRITE request changes the data, the same data will be available again on the next SREAD request.) RELEASE=NO is the default.

RELEASE=YES specifies that, after the SREAD request, the system is to release the storage that backed the data in the hiperspace.

If you code RELEASE=YES, do not code HSPALET.

,**RNSCODE=rsn-addr**

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

,**RETCODE=ret-addr**

Specifies the location where the system is to store the return code. The return code is also in GPR 15.

,**MF=S**

Specifies the standard form of the macro. This form generates code to place the parameters into an inline parameter list and invoke the service.

**ABEND codes**

HSPSERV might abnormally terminate with abend code X'01D'. See z/OS MVS System Codes for an explanation of abend code X'01D'.

 NUMRANGE=3, RANGLIST=(5)

 or

 NUMRANGE=3, RANGLIST=RANGADDR

AddrSp Loc  HiperLoc  Blocks

AddrSp Loc  HiperLoc  Blocks

AddrSp Loc  HiperLoc  Blocks

Register5  RANGADDR (fullword)

12Bytes
### Return and reason codes

When control returns from HSPSERV SREAD or HSPSERV SWRITE, GPR 15 (and ret-addr, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and rsn-addr, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

<table>
<thead>
<tr>
<th>Return Code</th>
<th>Reason Code</th>
<th>Meaning and Action</th>
</tr>
</thead>
</table>
| 00          | 00          | **Meaning:** HSPSERV completed successfully.  
**Action:** None. |
| 08          | xxyy05xx    | **Meaning:** System error. The system rejects the request. A hiperspace page is unavailable.  
**Action:** Record the return and reason code and supply it to the appropriate IBM support personnel. |
| 08          | xxyy06xx    | **Meaning:** System error. The system rejects the request. An address space page is unavailable.  
**Action:** Record the return and reason code and supply it to the appropriate IBM support personnel. |
| 0C          | xx006xx     | **Meaning:** System error. System failure due to environmental problems.  
**Action:** Record the return and reason code and supply it to the appropriate IBM support personnel. |

**Note:** yy is X’09’ for SREAD and X’0A’ for SWRITE.

### HSPSERV - List form

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

#### Syntax

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

```plaintext
name name

/bslash
HSPSERV
/bslash

PLISTVER=vernum

, MF=(L, list addr)

name: Symbol. Begin name in column 1.

b

One or more blanks must precede HSPSERV.

HSPSERV

b

One or more blanks must follow HSPSERV.

vernum: Parameter list version 0 or 1.

Default: Version that allows all specified parameters.

list addr: Symbol.
```
HSPSERV macro

\[\text{MF=(L, list addr, attr)}\]

\text{attr}: 1- to 60-character input string. \textbf{Default:} 0D

---

Parameters

Parameters for the list form of HSPSERV are as follows:

\textbf{PLISTVER=vernum}

Specifies the macro version associated with HSPSERV. PLISTVER is an optional parameter that determines which parameter list the system generates. Specify zero if you use parameters only from this group:

- MF
- NUMRANGE
- PLISTVER
- RANGLIST
- RELEASE
- RETCODE
- RSNCODE
- SREAD
- STOKEN
- SWRITE

If you use the HSFALET parameter, specify 1.

If you do not specify PLISTVER, the default is to allow all of the parameters you specify on the invocation to be processed.

\[\text{MF=(L, list addr)}\]

\[\text{MF=(L, list addr, attr)}\]

Specifies the list form of HSPSERV.

- \text{list-addr} is the address of the storage area for the parameter list.
- \text{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 \text{attr}, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

HSPSERV - Execute form

The execute form of the HSPSERV macro changes parameters in the control parameter list that the system created through the list form of the macro and performs the specified operation.

Syntax

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

\[\text{name} \quad \text{name: Symbol. Begin name in column 1.}\]

\[b\]

One or more blanks must precede HSPSERV.
One or more blanks must follow HPSERV.

SREAD
SWRITE

,STOKEN=stoken-addr
  stoken-addr: RX-type address or register (2) - (12).

,HSPELAT=alet-addr
  alet-addr: RX-type address or register (2) - (12).

,NUMRANGE=1
  Default: NUMRANGE=1.

,NUMRANGE=num-addr
  num-addr: RX-type address or register (2) - (12).

,RANGLIST=list-addr
  list-addr: RX-type address or register (2) - (12).

,RELEASE=NO
  Default: RELEASE=NO.

,RELEASE=YES

,RETCODE=ret-addr
  ret-addr: RX-type address or register (2) - (12).

,RSNCODE=rsn-addr
  rsn-addr: RX-type address or register (2) - (12).

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

,MF=(E,list-addr,NOCHECK)
  Default: COMPLETE.

Parameters

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

,MF=(E,list-addr,COMPLETE)
  list-addr specifies the area that the system uses to store the parameters.

,MF=(E,list-addr,NOCHECK)
  COMPLETE, which is the default, specifies that the system checks for required parameters and supplies the 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.

HPSERV - Modify form

Use the modify form of the HPSERV 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.
**HSPSERV macro**

**Syntax**

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

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

SREAD
SWRITE

,HSPALET=alet-addr  alet-addr: RX-type address or register (2) - (12).
,STOKEN=stoken-addr stoken-addr: RX-type address or register (2) - (12).
,NUMRANGE=1         Default: NUMRANGE=1.
,NUMRANGE=num-addr  num-addr: RX-type address or register (2) - (12).
,RANGLIST=list-addr list-addr: RX-type address or register (2) - (12).
,RELEASE=NO         Default: RELEASE=NO.
,RELEASE=YES

,RETCODE=ret-addr   ret-addr: RX-type address or register (2) - (12).
,RSNCODE=rsn-addr   rsn-addr: RX-type address or register (2) - (12).
,MF=(M,list-addr,COMPLETE) list-addr: RX-type address or register (2) - (12).
,MF=(M,list-addr,NOCHECK)  Default: COMPLETE.
```

**Parameters**

Parameters for the modify form of HSPSERV are described under the standard form of the macro with the following exceptions:

- \( ,MF=(M,\text{list-addr},\text{COMPLETE}) \)
- \( ,MF=(M,\text{list-addr},\text{NOCHECK}) \)

Specifies the modify form of the 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 the 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.
Appendix. Accessibility

Publications for this product are offered in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when using PDF files, you may view the information through the z/OS Internet Library web site or the z/OS Information Center. If you continue to experience problems, send an email to mhvrdfs@us.ibm.com or write to:

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. The major accessibility features in z/OS enable users to:

- Use assistive technologies such as screen readers and screen magnifier software
- Operate specific or equivalent features using only the keyboard
- Customize display attributes such as color, contrast, and font size.

Using assistive technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/OS. Consult the assistive technology documentation for specific information when using such products to access z/OS interfaces.

Keyboard navigation of the user interface

Users can access z/OS user interfaces using TSO/E or ISPF. Refer to z/OS TSO/E Primer, z/OS TSO/E User's Guide and z/OS ISPF User's Guide Vol I for information about accessing TSO/E and ISPF interfaces. These guides describe how to use TSO/E and ISPF, including the use of keyboard shortcuts or function keys (PF keys). Each guide includes the default settings for the PF keys and explains how to modify their functions.

z/OS information

z/OS information is accessible using screen readers with the Library Server versions of z/OS books in the Internet library at:

http://www.ibm.com/systems/z/os/zos/bkserv/
This information was developed for products and services offered in the U.S.A. or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

Site Counsel
IBM Corporation
2455 South Road
Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information
This information is intended to help the customer to code macros that are available to all assembler language programs. This information documents intended programming interfaces that allow the customer to write programs to obtain services of z/OS.

Policy for unsupported hardware
Various z/OS elements, such as DFSMS, HCD, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.
Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml.

UNIX is a registered trademark of The Open Group in the United States and other countries.
Index

Numerics
31-bit addressing mode
macros requiring expansion
  CALL 119
  ESTAE macro 563
  EVENTS 577

A
abend
  interrupting scheduled 563
ABEND macro 25
abnormal termination
  caused by failure to remove a subtask 459
  of a task 25
accessibility 629
contact IBM 629
features 629
  screen readers 631
addressing mode and the services 2
ALESERV macro 29
ALET qualification
  of parameters 3
AR () mode
  description 3
ASASYMBM service 43
ASC (address space control) mode
  defining 3
assistive technologies 629
ATTACH and ATTACHX macros 47

B
BLDMPB macro 67
BLSABDPL macro 71
BLSACBSP macro 75
BLSADSY macro 77
BLSAPCOE macro 79
BLSQFXL macro 81
BLSOMDEF macro 83
BLSOMFLD macro 87
BLSQSHDR macro 99
BLSRDRPX macro 101
BLSRESSY macro 103
BLSRNAMP macro 105
BLSRPRED macro 107
BLSRPWHS macro 109
BLSRSASY macro 111
BLSRXSMSP macro 113
BLSRXSSP macro 115
BLSUPPR2 macro 117

C
CALL macro 119
callable service
  coding 16
  cell pool
    creating 165
    deleting 165
    obtaining 165
    placing start/end addresses of cell pool extents in a
    buffer 165
    returning 165
    services 165
CHAP macro 125
CnzConv macro 129
CNZMXURF macro 141
CNZTRKR macro 145
coding the callable services 16
coding the macros 13
completed ECB
  list 580
compression
  services 187
console ID
  locating 141
contention queue element create service
  parameter list
    initialization 80
continuation line 15
control
  passing between control sections 119
control block
  specifying a formatting model field 87
control block format model
  defining 83
CONVCON macro 149
CONVTOD macro 157
CPOOL macro 165
CPU ID
  retrieving 375
CPU timer
  obtaining value 177
CPUTIMER macro 177
  relationship to STIMER macro 177
CSRC4ACT callable service 193
CSRC4BLD callable service 197
CSRC4CON callable service 201
CSRC4DAC callable service 205
CSRC4DIS callable service 209
CSRC4EXP callable service 213
CSRC4FR1 callable service 223
CSRC4FR2 callable service 227
CSRC4FRE callable service 219
CSRC4GET callable service 231
CSRC4GT1 callable service 235
CSRC4GT2 callable service 239
CSRC4QCL callable service 243
CSRC4QEX callable service 247
CSRC4QPL callable service 251
CSRC4RF1 callable service 259
CSRC4RFR callable service 255
CSRC4RG1 callable service 267
CSRC4RGT callable service 263

© Copyright IBM Corp. 1988, 2012
<table>
<thead>
<tr>
<th>Service</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSRCESRV macro</td>
<td>181</td>
</tr>
<tr>
<td>CSREVW callable service</td>
<td>271</td>
</tr>
<tr>
<td>CSRIDAC callable service</td>
<td>275</td>
</tr>
<tr>
<td>CSR16J callable service</td>
<td>281</td>
</tr>
<tr>
<td>CSRPACT callable service</td>
<td>285</td>
</tr>
<tr>
<td>CSRPBLD callable service</td>
<td>289</td>
</tr>
<tr>
<td>CSRPCON callable service</td>
<td>293</td>
</tr>
<tr>
<td>CSRPDAC callable service</td>
<td>297</td>
</tr>
<tr>
<td>CSRPDIS callable service</td>
<td>301</td>
</tr>
<tr>
<td>CSRPXEXP callable service</td>
<td>305</td>
</tr>
<tr>
<td>CSRPFR1 callable service</td>
<td>315</td>
</tr>
<tr>
<td>CSRPFR2 callable service</td>
<td>319</td>
</tr>
<tr>
<td>CSRPFRF callable service</td>
<td>311</td>
</tr>
<tr>
<td>CSRPGET callable service</td>
<td>323</td>
</tr>
<tr>
<td>CSRPOT1 callable service</td>
<td>327</td>
</tr>
<tr>
<td>CSRPOT2 callable service</td>
<td>331</td>
</tr>
<tr>
<td>CSRPOCL callable service</td>
<td>335</td>
</tr>
<tr>
<td>CSRPOQEX callable service</td>
<td>339</td>
</tr>
<tr>
<td>CSRPOPL callable service</td>
<td>343</td>
</tr>
<tr>
<td>CSRPRFR callable service</td>
<td>347</td>
</tr>
<tr>
<td>CSRPRFR1 callable service</td>
<td>351</td>
</tr>
<tr>
<td>CSRPRGT callable service</td>
<td>355</td>
</tr>
<tr>
<td>CSRPRGT1 callable service</td>
<td>359</td>
</tr>
<tr>
<td>CSRREFR callable service</td>
<td>363</td>
</tr>
<tr>
<td>CSRSAVE callable service</td>
<td>367</td>
</tr>
<tr>
<td>CSRSCOT callable service</td>
<td>371</td>
</tr>
<tr>
<td>CSRSI</td>
<td>375</td>
</tr>
<tr>
<td>CSRUNIC macro</td>
<td>393</td>
</tr>
<tr>
<td>CSRVIEW callable service</td>
<td>403</td>
</tr>
<tr>
<td>CSVAPF macro</td>
<td>407</td>
</tr>
<tr>
<td>CSVINFO macro</td>
<td>419</td>
</tr>
<tr>
<td>CSVQUERY macro</td>
<td>431</td>
</tr>
</tbody>
</table>

D

data
  compressing 187
  expanding 187
delete macro
  relationship to LOAD macro 445
DELETE macro 445
DEQ macro 449
DETACH macro 459
dispatching priority
  changing 125
DIV macro 463
  use with IARVSERV macro 464
DOM macro 487
DSPSERV macro
  for data spaces 491
  for hiperspaces 505
  limitation with IARVSERV macro 494
dumping service
  defining a control block format model 83
  formatting routine parameters 71, 103
  specifying a formatting model field 87

E

ECB (event control block)
  initializing 580

ECB (event control block) (continued)
  list of completed 580
EDT (eligible device table)
  obtaining information 519
EDTINFO macro 519
ENQ macro 539
ESPIE environment
  deleting 553
  determining 553
  establishing 553
ESPIE macro 553
ESTAE and ESTAEX macros 563
EVENTS macro 577
  events table
    creating 577, 579
    deleting 577, 580
    format 580
  exit routine
    end-of-task 25

F

FREEMAIN macro 585

G

GETMAIN macro 593
global serialization queue
  extracting information 607
GQSCAN macro 607

H

hiperspace
  reading 619
  writing 619
HSPSERV macro 619

K

keyboard
  navigation 629
  PF keys 629
  shortcut keys 629

L

list
  of completed ECBs 580
LOAD macro
  relationship to DELETE macro 445
load module
  deleting 445
  releasing control 445

M

macro
  coding 13
  forms 11
macro (continued)
  level
    selecting 1
  sample 14
  selecting level 1
  user parameter, passing 4
  X-macros
    using 11
mainframe
  education xii
message
  deleting 487

N
  navigation
    keyboard 629
  Notices 631

O
  operator message
    deleting 487

P
  parameter list
    used in EVENTS processing 580
  processor ID
    retrieving 375

R
  recovery routine
    establishing an ESTAE-type 563
  requesting processing
    group of instructions 393

S
  screen readers
    accessibility 631
  serially reusable resource
    releasing 449
    requesting control 539
  service
    ALET qualification 3
    summary 17
  services
    addressing mode 2
    ASC mode
      defining 3
      using 1
  shortcut keys 629
  STIMER macro
    relationship to CPUTIMER macro 177
  subtask
    detaching 459
  system information service
    retrieve system information 375

T
  task
    creating 47
  TOD (time-of-day) clock
    converting value 157

U
  user interface
    ISPF 629
    TSO/E 629
  user parameter
    passing 4

V
  virtual storage
    allocating 585, 593
    releasing 445

X
  X-macros
    using 11

Z
  z/OS Basic Skills information center xii