z/OS
2.4

MVS Programming: Assembler Services Reference, Volume 1 (ABE-HSP)

SA23-1369-40

Note

Before using this information and the product it supports, read the information in "Notices" on page 723.

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

Last updated: 2021-04-05

Contents

Figures

Tables

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. For more information about assembler language programming, see High Level Assembler and Toolkit Feature in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSENW6).

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

How to use this 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*.

z/OS information

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

When possible, this information uses cross document links that go directly to the topic in reference using shortened versions of the document title. For complete titles and order numbers of the documents for all products that are part of z/OS, see *z/OS Information Roadmap*.

To find the complete z/OS® library, go to IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

How to send your comments to IBM

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

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

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

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

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

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

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

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS MVS Assembler Services Reference ABE-HSP, SA23-1369-50
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive authority to use or distribute the comments in any way appropriate without incurring any obligation to you.

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

If you have a technical problem

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

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

Summary of changes

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

Summary of changes for z/OS Version 2 Release 4

New

The following information has been added in this publication:

July 2020 refresh

- The BCFXCRYP service has been added in Chapter 7, "BCFXCRYP — Interface to invoke the protected key encryption service," on page 77.

Prior to July 2020 refresh

- The ASAXWC service has been added in Chapter 4, "ASAXWC — Wildcard service," on page 41.
- RMODE 64-bit capabilities have been added to Chapter 86, "DELETE — Relinquish control of a load module," on page 471.

Changed

March 2021 refresh

- For the ENQ service, the description of the RET=USE parameter has been updated. See Parameters.
- The syntax diagram for the GETMAIN service is updated to clarify the use of the LOC parameter. See Chapter 100, "GETMAIN — Allocate virtual storage," on page 647.

July 2020 refresh

- The description and programming requirements sections are updated to reflect the L16J and L16J1 data areas as defined in macro CSRYL16J. For more information, see Chapter 57, "CSRL16J — Transfer control with all registers intact," on page 305.
- The description of the CONVTOD service is updated. See "Description" on page 171.

Summary of changes for z/OS Version 2 Release 3

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

New

- New macro Chapter 99, "FXECNTRL — Maintain Function Exploitation and Enablement," on page 629.
- New reason codes have been added for the DIV service in Table 16 on page 497.

Changed

- The ,ADDRENV=SAME description of the ATTACHX macro was enhanced for clarity.
- The descriptions for EVENTADDR, EVENTPSW16, and EVENTPSW8 were updated in Chapter 102, "GTZTRACK macro — GTZ Track," on page 679.

Summary of changes for z/OS Version 2 Release 2

The following information is new, changed, or deleted in z/OS MVS Programming: Assembler Services Reference ABE-HSP in z/OS Version 2 Release 2 (V2R2).

New

- New ASASYMBF macro substituting text for system symbols added to Chapter 5, “ASASYMBM and ASASYMBF — Substitute text for symbols,” on page 49.
- The PAGEFRAMESIZE parameter has been added in Chapter 91, “DSPSERV — Create, delete, and control data spaces,” on page 519.

Changed

- Updates to information for macro ASASYMBM substituting text for system symbols added to z/OS MVS Programming: Assembler Services Reference ABE-HSP.
- Information about the PREFIX parameter has been updated in Chapter 14, “BLSQMDEF — Define a control block format model,” on page 97.
- Information about the syntax, parameters, and examples has been updated in Chapter 15, “BLSQMFLD — Specify a formatting model field,” on page 103.

Chapter 1. Using the services

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

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

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

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

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

Compatibility of MVS macros

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

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

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

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

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

Addressing mode (AMODE)

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

In general:

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

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

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

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

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

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

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

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

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

For information about AMODE 64 and the 64-bit GPR, see *z/OS MVS Programming: 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 for more information. Table 1 lists the macros that check the global symbol.

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

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

ALET qualification

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

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

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

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

The only ALETs that MVS services typically accept are:

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

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

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

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

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

User parameters

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

Note:

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

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

Table 1. Passing User Parameters in AR Mode

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

When an AR mode caller who is using a 4-bytes-per-entry parameter list passes ALET-qualified addresses to the called program through PARAM,VL=1 on the ATTACH/ATTACHX, 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.

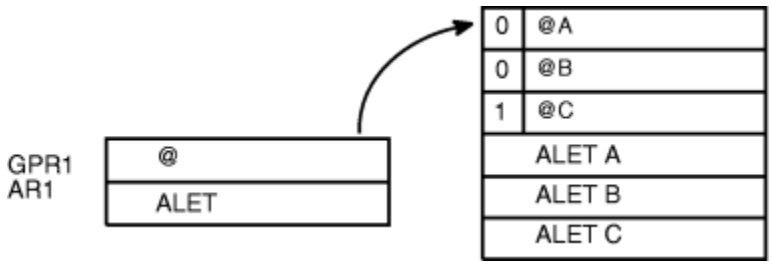


Figure 1. Sample User Parameter List for Callers in AR Mode

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

Telling the system about the execution environment

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

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

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

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

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

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

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

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

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

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

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

Specifying a macro version number

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

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

How to request a macro version using PLISTVER

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

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

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

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

Hints for using PLISTVER

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

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

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

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

Register use

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

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

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

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

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

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

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

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

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

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

Handling return codes and reason codes

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

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

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

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

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

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

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

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

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

Handling program errors

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

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

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

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

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

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

Handling environmental and system errors

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

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

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

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

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

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

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

Using X-macros

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

The X-macros described in this information are:

- ATTACHX
- ESTAEX
- LINKX
- SNAPX
- SYNCHX
- XCTLX

The only way these macros know that a caller is in AR mode is by checking the global symbol that the SYSSTATE macro sets. Each of these macros (and corresponding non-X-macro) checks the symbol. If

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

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

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

- Callers in primary mode can invoke either macro.

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

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

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

Macro forms

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

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

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

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

Conventional list form macros

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

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

5. Invoke the macro by issuing the execute form of the macro. Use the address parameter of the execute form to reference the parameter list in the virtual storage area that you obtained.

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

Alternative list form macros

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

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

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

Coding the macros

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

The syntax tables assume that the standard begin, end, and continue columns are used. Thus, column 1 is assumed as the begin column. To change the begin, end, and continue columns, use the ICTL instruction to establish the coding format you want to use. If you do not use ICTL, the assembler recognizes the standard columns. For more information about coding the ICTL instruction, see High Level Assembler and Toolkit Feature in IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSENW6).

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

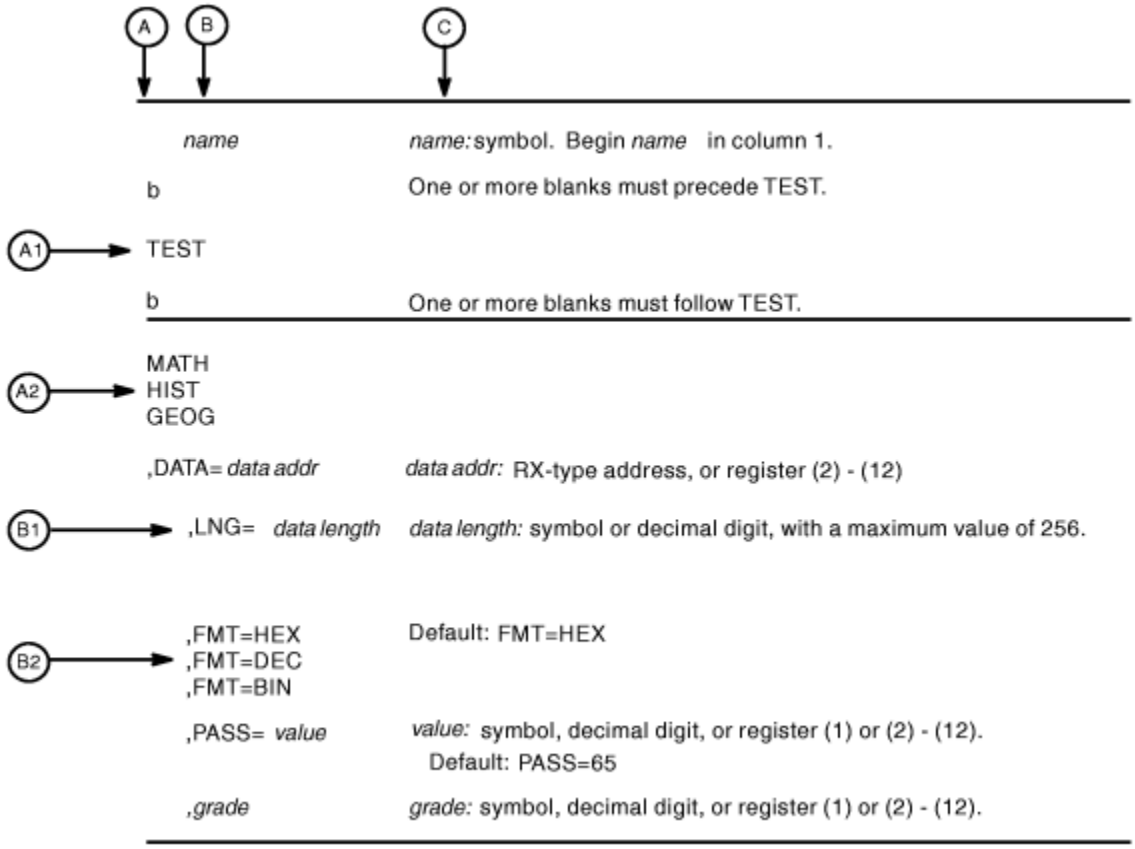


Figure 2. Sample tabular syntax diagram for the TEST macro

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

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

Variable
Classification

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

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

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

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

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

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

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

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

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

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

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

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

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

Continuation lines

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

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

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

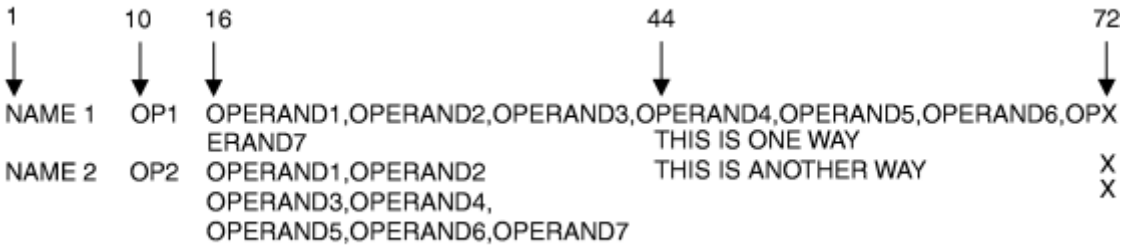


Figure 3. Continuation Coding

Coding the callable services

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

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

The syntax diagram for the sample callable service SCORE:

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

Considerations for coding callable services are:

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

Including equate (EQU) statements

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

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

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

Link-editing linkage-assist routines

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

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

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

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

Service summary

Table 1 lists services described in the following:

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

For each service, the table indicates:

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

Note:

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

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

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

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

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

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

Table 3. Service Summary

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
ABEND	Yes	Yes	Yes	Yes
ALESERV	Yes	Yes	No	No
ASASYMBM	No	No	Yes	No
ATTACH	Yes (See note 1)	No	Yes	No
ATTACHX	Yes	No	Yes	Yes
BLDMPB	Yes	Yes	No	No
BLSABDPL	Yes	Yes	N/A	No
BLSACBSP	Yes	Yes	N/A	No
BLSADSY	Yes	Yes	N/A	No
BLSAPCQE	Yes	Yes	N/A	No
BLSQFXL	Yes	Yes	N/A	No
BLSQMDEF	Yes	Yes	N/A	No
BLSQMFLD	Yes	Yes	N/A	No
BLSQSHDR	Yes	Yes	N/A	No
BLSRDRPX	Yes	Yes	N/A	No
BLSRESSY	Yes	Yes	N/A	No
BLSRNAMP	Yes	Yes	N/A	No
BLSRPRD	Yes	Yes	N/A	No
BLSRPWHS	Yes	Yes	N/A	No
BLSRSASY	Yes	Yes	N/A	No
BLSRXMSP	Yes	Yes	N/A	No
BLSRXSSP	Yes	Yes	N/A	No
BLSUPPR2	Yes	Yes	N/A	No
CALL	Yes	Yes	Yes	Yes
CHAP	No	No	No	Yes
CNZCONV	Yes	Yes	No	Yes
CNZTRKR	No	Yes	No	No
CONVCON	No	Yes	No	No
CONVTOD	Yes	Yes	No	No
CPOOL	No	Yes	Yes	No
CPUTIMER	No	Yes	Yes	No
CSRCESRV	Yes	Yes	No	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
CSRCMPSC	Yes	Yes	Yes	No
CSREVW	No	No	N/A	No
CSRIDAC	No	No	N/A	No
CSRL16J	No	No	N/A	No
CSRPACT	Yes	Yes	N/A	No
CSRPBLD	Yes	Yes	N/A	No
CSRPCON	Yes	Yes	N/A	No
CSRPDAC	Yes	Yes	N/A	No
CSRPDIS	Yes	Yes	N/A	No
CSRPEXP	Yes	Yes	N/A	No
CSRPFRE	Yes	Yes	N/A	No
CSRPFR1	Yes	Yes	N/A	No
CSRPGET	Yes	Yes	N/A	No
CSRPGT1	Yes	Yes	N/A	No
CSRPQCL	Yes	Yes	N/A	No
CSRPQEX	Yes	Yes	N/A	No
CSRPQPL	Yes	Yes	N/A	No
CSRPRFR	Yes	Yes	N/A	No
CSRPRFR1	Yes	Yes	N/A	No
CSRPRGT	Yes	Yes	N/A	No
CSRPRGT1	Yes	Yes	N/A	No
CSRREFR	No	No	N/A	No
CSRSAVE	No	No	N/A	No
CSRSCOT	No	No	N/A	No
CSRSI	No	Yes	No	No
CSRUNIC	Yes	Yes	No	No
CSRVIEW	No	No	N/A	No
CSVAPF	Yes (See note 7)	Yes	Yes	No
CSVINFO	No	No	No	No
CSVQUERY	Yes	Yes	Yes	Yes
DELETE	No	No	No	Yes
DEQ	No	No	No	Yes
DETACH	Yes	No	Yes	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
DIV	Yes	No	Yes	No
DOM	No	No	No	Yes
DSPSERV	Yes	Yes	Yes	Yes
EDTINFO	Yes	Yes	Yes	No
ENQ	No	No	No	Yes
ESPIE	No	No	No	Yes
ESTAE (See note 2)	No	No	Yes	No
ESTAEX	Yes	Yes	Yes	Yes
EVENTS	No	No	No	No
FREEMAIN	No (See note 3)	Yes	Yes	Yes
GETMAIN	No (See note 3)	Yes	Yes	Yes
GQSCAN	No	Yes	No	No
HSPSERV	Yes	Yes (See note 4)	(See note 5)	No
IARCP64	Yes	Yes	Yes	Yes
IARR2V	Yes	Yes	No	Yes
IARST64	Yes	Yes	Yes	Yes
IARVSERV	Yes	Yes	Yes	No
IARV64	Yes	Yes	Yes	Yes
IDENTIFY	No	No	No	Yes
IEAARR	Yes	Yes	Yes	No
IEABRC	Yes	Yes	N/A	No
IEAINTKN	Yes	Yes	Yes	No
IEALSQRY	Yes	Yes	Yes	No
IEAMETR	Yes	Yes	Yes	No
IEANTCR	Yes	Yes	N/A	No
IEANTDL	Yes	Yes	N/A	No
IEANTRT	Yes	Yes	N/A	No
IEATDUMP	Yes	No	Yes	No
IEATXDC	Yes	Yes	Yes	Yes
IEAVAPE	No	Yes	No	No
IEAVAPE2	No	Yes	No	No
IEAVDPE	No	Yes	No	No
IEAVDPE2	No	Yes	No	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
IEAVPSE	No	Yes	No	No
IEAVPSE2	No	Yes	No	No
IEAVRLS	No	Yes	No	No
IEAVRLS2	No	Yes	No	No
IEAVRPI	No	Yes	No	No
IEAVRPI2	No	Yes	No	No
IEAVTPE	No	Yes	No	No
IEAVXFR	No	Yes	No	No
IEAVXFR2	No	Yes	No	No
IEA4APE	No	Yes	No	Yes
IEA4APE2	No	Yes	No	Yes
IEA4DPE	No	Yes	No	Yes
IEA4DPE2	No	Yes	No	Yes
IEA4PSE	No	Yes	No	Yes
IEA4PSE2	No	Yes	No	Yes
IEA4RLS	No	Yes	No	Yes
IEA4RLS2	No	Yes	No	Yes
IEA4RPI	No	Yes	No	Yes
IEA4RPI2	No	Yes	No	Yes
IEA4TPE	No	Yes	No	Yes
IEA4XFR	No	Yes	No	Yes
IEA4XFR2	No	Yes	No	Yes
IEFDDSRV	Yes	Yes	No	No
IEFSSI	Yes	No	No	No
IOCINFO	Yes	Yes	Yes	No
IOSCHPD	Yes	Yes	Yes	No
ITZEVENT	No	Yes	No	No
ITZQUERY	No	Yes	No	No
IXGBRWSE	Yes	Yes	Yes	Yes
IXGCONN	Yes	Yes	Yes	Yes
IXGDELET	Yes	Yes	Yes	Yes
IXGIMPRT	Yes	Yes	Yes	Yes
IXGINVNT	Yes	Yes	Yes	Yes

Table 3. Service Summary (continued)

Service	Can be issued in AR ASC mode	Can be issued in cross memory mode	Checks SYSSTATE	Can be issued in 64-bit AMODE
IXGOFFLD	Yes	Yes	Yes	Yes
IXGQUERY	Yes	Yes	Yes	Yes
IXGUPDAT	Yes	Yes	Yes	Yes
IXGWRITE	Yes	Yes	Yes	Yes
LINK	Yes (See note 1)	No	Yes	No
LINKX	Yes	No	Yes	Yes
LOAD	Yes	No	No	Yes
LSEXPAND	Yes	No	No	No
PGLOAD	No	No	No	No
PGOUT	No	No	No	No
PGRLSE	No	No	No	No
PGSER	No	No	No	Yes
POST	No	Yes	No	Yes
QRYLANG	Yes	Yes	No	No
REFPAT	Yes	No	Yes	No
RESERVE	No	No	No	Yes
RETURN	No	No	No	No
SAVE	No	No	No	No
SETRP	Yes	Yes	Yes	Yes
SNAP	Yes (See note 1)	No	Yes	No
SNAPX	Yes	No	Yes	No
SPIE	No	No	No	No
SPLEVEL	Yes	Yes	No	No
STAE	No	No	No	No
STATUS	Yes	Yes	No	No
STCKCONV	Yes	Yes	No	No
STCKSYNC	Yes	Yes	Yes	No
STIMER	No	No	No	Yes
STIMERM	No	No	No	Yes
STORAGE	Yes	Yes	No	Yes
SYMRBLD	Yes	Yes	Yes	No
SYMREC	No	Yes	Yes	No
SYNCH	Yes (See note 1)	No	Yes	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
SYNCHX	Yes	No	Yes	Yes
SYSSTATE	Yes	Yes	No	No
TCBTOKEN	Yes	Yes	No	No
TESTART	Yes	Yes	No	No
TIME	Yes (See note 6)	Yes (See note 6)	No	Yes
TIMEUSED	Yes	Yes	No	Yes
TRANMSG	Yes	Yes	No	No
TTIMER	No	No	No	Yes
UCBDEVN	No	No	No	No
UCBINFO	Yes	Yes	Yes	No
UCBSCAN	Yes	Yes	Yes	No
UPDTMPB	Yes	Yes	No	No
VRADATA	Yes	Yes	Yes	No
WAIT	No	Yes	No	Yes
WTL	No	No	No	No
WTO	No	No	No	Yes
WTOR	No	No	No	Yes
XCTL	Yes (See note 1)	Yes	Yes	Yes
XCTLX	Yes	Yes	Yes	No

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

Description

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:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- 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.

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ABEND.
ABEND	
ƀ	One or more blanks must follow ABEND.
comp code	*comp code*: Symbol, decimal or hexadecimal digit, or register (1) or (2) - (12). **Value range**: 0 - 4095
,REASON=*reason code*	*reason code:* Symbol, decimal or hexadecimal number, or register (2) - (12).
,DUMP	
,,STEP	
,,,*code type*	*code type:* USER or SYSTEM. **Default**: *code type* = USER.
,DUMP,STEP	
,DUMP,,*code type*	
,,STEP,*code type*	
,DUMP,STEP,*code type*	
,DUMP,DUMPOPT=*parm list addr*	*parm list addr:* RX-type address, or register (2) - (12).

Syntax	Description
,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'dd' format to distinguish the hexadecimal from decimal.

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

,DUMP
,,STEP
,,,*code type*
,DUMP,STEP
,DUMP,,*code type*
,,STEP,*code type*
,DUMP,STEP,*code type*
,DUMP,DUMPOPT=*parm list addr*
,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 UNIX system services, the system writes a *core dump*, which is a SYSMDUMP to a z/OS UNIX file, for errors following an exec or fork() function when the original address space had a SYSMDUMP DD statement. For more information, see *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.

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

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.

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 *z/OS MVS Programming: Assembler Services Guide*. That book contains many examples of using ALESERV.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- 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
- 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:

Register
Contents

0
Reason code associated with the return code for SEARCH and EXTRACT requests; otherwise, used as a work register by the system

1
Address of the ALESERV parameter list

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0
Used as a work register by the system

1
ALET for the parameter list

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ALESERV.
ALESERV	
ƀ	One or more blanks must follow ALESERV.
	Valid parameters (required parameters are underlined):
ADD	AL, STOKEN, ACCESS, ALET, CHKPT, RELATED
ADDPASN	ALET, CHKPT, RELATED
DELETE	ALET, RELATED
EXTRACT	ALET, STOKEN, RELATED
SEARCH	ALET, STOKEN, AL, RELATED
EXTRACTH	STOKEN, RELATED
,ACCESS=PUBLIC	**Default**: ACCESS=PUBLIC
,ACCESS=PRIVATE	
,AL=WORKUNIT	**Default**: AL=WORKUNIT
,AL=PASN	
,ALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12). **Note:** If you specify register notation, the register contains the ALET, rather than the address of the ALET.
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address.
,CHKPT=FAIL	**Default**: CHKPT=FAIL
,CHKPT=IGNORE	
,RELATED=*any-value*	

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 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.

,STOKEN=*stoken-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.

,CHKPT=FAIL
,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 *z/OS MVS Programming: Assembler Services Guide* for more information on using checkpoints with data spaces and hiperspaces.

,RELATED=*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.

Hexadecimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Meaning and Action
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. **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.
18	**Meaning**: 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. **Action**: Change the request to add the data space as SCOPE=SINGLE or change your program to run in supervisor state or key 0 - 7.
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 a PASN-AL ALET. **Action**: Verify that AR 1 contains either an ALET of 0 or the ALET for the caller's DU-AL.
38	**Meaning**: Program error. The input STOKEN is not valid. **Action**: Verify that the specified STOKEN is a valid STOKEN.
4C	**Meaning**: Program or environmental error. The space represented by the input STOKEN is not valid for cross memory access. **Action**: None required. However, you may want to 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.
54	**Meaning**: Program error. The caller tried to add a data space or hiperspace to an access list as a private entry. **Action**: Specify ACCESS=PUBLIC instead of ACCESS=PRIVATE.
5C	**Meaning**: Program error. The caller tried to add a data space or a hiperspace to an access list without proper authority. **Action**: Correct your program to specify STOKENs for spaces for which your program is authorized.
60	**Meaning**: System error. An unexpected error occurred. The request was not completed. **Action**: Retry the request.

Hexadecimal Return Code	Meaning and Action
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 ADD request. **Action**: Determine the cause of the error that preceded the ALESERV ADD request. Correct the error and rerun the program.
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 HSPSERV 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:

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: ALESERV ADDPASN has completed successfully. **Action**: None.
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:

Hexadecimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Meaning and Action
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.
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:

Hexadecimal Return Code	Meaning and Action
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:

Hexadecimal Return Code	Meaning and Action
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:

Hexadecimal Return Code	Meaning and Action
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 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ALESERV.
ALESERV	
␣	One or more blanks must follow ALESERV.
MF=L	
,RELATED=*any-value*	

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:

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

Syntax	Description
ƀ	One or more blanks must precede ALESERV.
ALESERV	
ƀ	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, STOKEN, 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*	*alet-addr*: RX-type address or register (2) - (12). **Note:** If you specify register notation, the register contains the ALET, rather than the address of the ALET.
,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.

Chapter 4. ASAXWC — Wildcard service

Description

ASAXWC is an executable macro that does wildcard match checking, comparing an input pattern string (which can contain wildcard characters) to an input string.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. 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:	None
Control parameters:	Control parameters must be in the primary address space.

Programming requirements

None.

Restrictions

No EUT FRRs may be in effect.

Input register information

None.

Output register information

None.

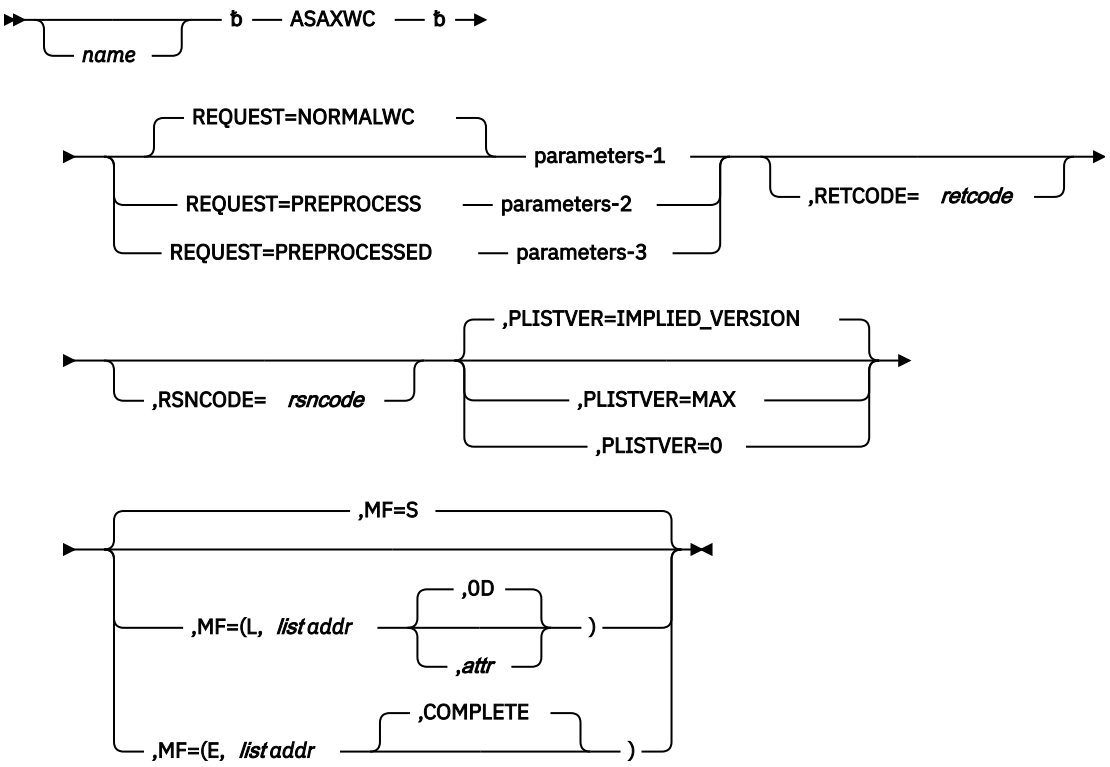
Performance implications

- This macro should not be used in performance-sensitive code.
- For REQUEST=NORMALWC, use of the WORKAREA keyword will result in better performance.
- When you intend to use the same pattern against multiple strings, use of REQUEST=PREPROCESS followed by multiple REQUEST=PREPROCESSED calls will result in better performance than the corresponding number of REQUEST=NORMALWC calls.

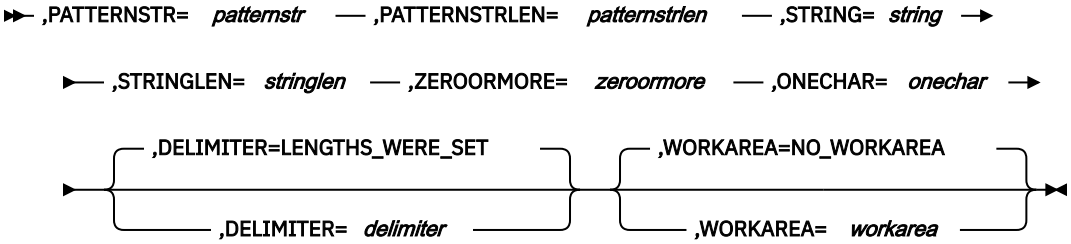
Syntax

The ASAXWC service has the following syntax:

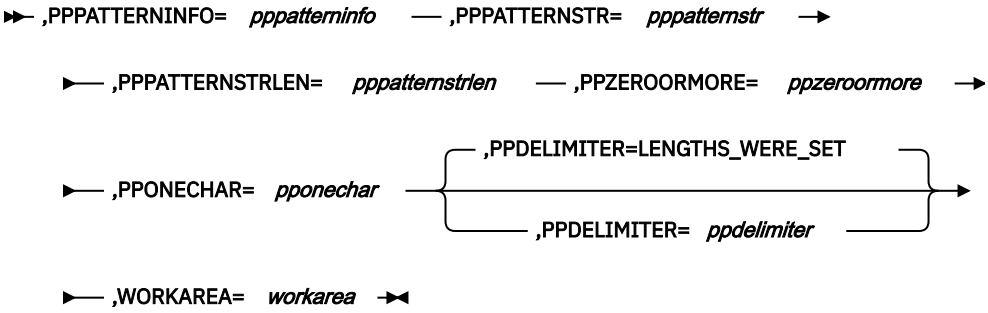
main diagram



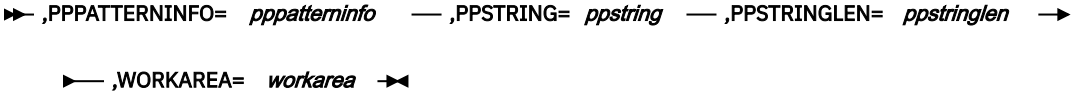
parameters-1



parameters-2



parameters-3



Parameters

The parameters are explained as follows:

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

,DELIMITER=*delimiter*
,DELIMITER=LENGTHS_WERE_SET
When REQUEST=NORMALWC is specified, an optional input parameter, field that specifies the character that indicates the end of the input (either PatternStr or String). The normal character is a blank. If the specified lengths are already correct, do not specify this keyword. The default is LENGTHS_WERE_SET.

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

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
An optional input parameter that specifies the macro form.

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

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

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

,list addr
The name of a storage area to contain the parameters. For MF=S and MF=E, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

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

,ONECHAR=*onechar*
When REQUEST=NORMALWC is specified, a required input parameter, field that specifies the wildcard character that represents exactly one character. The normal character is "?".

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

,PATTERNSTR=*patternstr*
When REQUEST=NORMALWC is specified, a required input parameter, field containing the wildcard pattern string.

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

,PATTERNSTRLEN=*patternstrlen*
When REQUEST=NORMALWC is specified, a required input parameter, field containing the length of the wildcard pattern string. The length actually used is the length up to the first occurrence of the delimiter character, or this length if the delimiter character is not present.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0

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

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,PPDELIMITER=*ppdelimiter*
,PPDELIMITER=LENGTHS_WERE_SET

When REQUEST=PREPROCESS is specified, an optional input parameter, field that specifies the character that indicates the end of the input (either PatternStr or String). The normal character is a blank. If the specified lengths are already correct, do not specify this keyword. The default is LENGTHS_WERE_SET.

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

,PPONECHAR=*pponechar*

When REQUEST=PREPROCESS is specified, a required input parameter, field that specifies the wildcard character that represents exactly one character. The normal character is "?".

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

,PPPATTERNINFO=*pppatterninfo*

When REQUEST=PREPROCESS is specified, a required output parameter, field that is to contain the preprocessed pattern information.

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

,PPPATTERNINFO=*pppatterninfo*

When REQUEST=PREPROCESSED is specified, a required input parameter, field that contains the preprocessed pattern information from a preceding REQUEST=PREPROCESS call. The pattern matching will use the same zero or more character symbol, one character symbol, and delimiter symbol identified on that REQUEST=PREPROCESS call.

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

,PPPATTERNSTR=*pppatternstr*

When REQUEST=PREPROCESS is specified, a required input parameter, field containing the wildcard pattern string.

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

,PPPATTERNSTRLEN=*pppatternstrlen*
When REQUEST=PREPROCESS is specified, a required input parameter, field containing the length of the wildcard pattern string. The length actually used is the length up to the first occurrence of the delimiter character, or this length if the delimiter character is not present.

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

,PPSTRING=*ppstring*
When REQUEST=PREPROCESSED is specified, a required input parameter, field containing the string to match against.

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

,PPSTRINGLEN=*ppstringlen*
When REQUEST=PREPROCESSED is specified, a required input parameter, field containing the length of the string to match against. The length actually used is the length up to the first occurrence of the delimiter character, or this length if the delimiter character is not present.

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

,PPZEROORMORE=*ppzeroormore*
When REQUEST=PREPROCESS is specified, a required input parameter, field that specifies the wildcard character that represents zero or more characters. The normal character is "*".

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

REQUEST=NORMALWC
REQUEST=PREPROCESS
REQUEST=PREPROCESSED
An optional parameter that indicates the type of request. The default is REQUEST=NORMALWC.

REQUEST=NORMALWC
Indicates to perform normal WC processing.

REQUEST=PREPROCESS
Indicates to preprocess a pattern string. The output from this function (via the PatternInfo keyword) is provided on a subsequent REQUEST=PREPROCESSED call.

REQUEST=PREPROCESSED
Indicates to use a preprocessed pattern string to do the wildcarding.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,STRING=*string*
When REQUEST=NORMALWC is specified, a required input parameter, field containing the string to match against.

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

,STRINGLEN=*stringlen*
When REQUEST=NORMALWC is specified, a required input parameter, field containing the length of the string to match against. The length actually used is the length up to the first occurrence of the delimiter character, or this length if the delimiter character is not present.

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

,WORKAREA=*workarea*
,WORKAREA=NO_WORKAREA
When REQUEST=NORMALWC is specified, an optional input parameter, field that specifies a doubleword-aligned area that the wildcard service can use. This will result in a call to the wildcard service entry that does no GETMAINs or FREEMAINs. While you cannot specify WORKAREA=(0), you can place the address of the workarea into register 0 and specify register 0 by a symbolic name (for instance, WORKAREA=(symbol) where symbol is equated to 0). Using WORKAREA=(R0) will result in the most efficient code being generated. Do not specify WORKAREA on the modify form. The default is NO_WORKAREA.

To code: Specify the RS-type address, or address in register (2) - (12), of a 256-character field.

,WORKAREA=*workarea*
When REQUEST=PREPROCESS is specified, a required input parameter, field that specifies a doubleword-aligned area that the wildcard service can use. While you cannot specify WORKAREA=(0), you can place the address of the workarea into register 0 and specify register 0 by a symbolic name (for instance, WORKAREA=(symbol) where symbol is equated to 0). Using WORKAREA=(R0) will result in the most efficient code being generated. Do not specify WORKAREA on the modify form. Do not specify WORKAREA=NO_WORKAREA.

To code: Specify the RS-type address, or address in register (2) - (12), of a 256-character field.

,WORKAREA=*workarea*
When REQUEST=PREPROCESSED is specified, a required input parameter, field that specifies a doubleword-aligned area that the wildcard service can use. While you cannot specify WORKAREA=(0), you can place the address of the workarea into register 0 and specify register 0 by a symbolic name (for instance, WORKAREA=(symbol) where symbol is equated to 0). Using WORKAREA=(R0) will result in the most efficient code being generated. Do not specify WORKAREA on the modify form. Do not specify WORKAREA=NO_WORKAREA.

To code: Specify the RS-type address, or address in register (2) - (12), of a 256-character field.

,ZEROORMORE=*zeroormore*
When REQUEST=NORMALWC is specified, a required input parameter, field that specifies the wildcard character that represents zero or more characters. The normal character is "*".

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

ABEND codes

None.

Return and reason codes

When the ASAXWC macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

The following table identifies the hexadecimal return and reason codes.

Table 4. Return and reason codes for the ASAXWC macro

Return code	Reason code	meaning and action
0	–	Match.
8	–	No match.

Example

```
     ASAXWC    PATTERNSTR=PS,      *
               PATTERNSTRLEN=PSL,  *
               STRING=S,           *
               STRINGLEN=SL,       *
               ZEROORMORE=Z,       *
               ONECHAR=O,          *
               DELIMITER=D,        *
               RETCODE=R,          *
               MF=(E,MYLIST)
PS       DC    C'HE?L*'
PSL      DC    A(L'PS)
S        DC    C'HELLO'
SL       DC    A(L'S)
Z        DC    CL1'*'
O        DC    CL1'?'
D        DC    CL1' '
DYNAREA  DSECT
R        DS    F
     ASAXWC    MF=(L,MYLIST)
```

Chapter 5. ASASYMBM and ASASYMBF — Substitute text for symbols

Description

Note: ASASYMBM and ASASYMBF are linkable system services.

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

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

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

The two services differ as follows:

- ASASYMBM obtains and releases dynamic storage for use by the symbol substitution service. Input is mapped using either the SYMBP or SYMBFP DSECT of the ASASYMBP mapping macro.
- ASASYMBF allows callers to pass in a 1024 byte work area for the symbol substitution service to use as dynamic storage, so that there is no need to obtain (and release) storage for that purpose. For ASASYMBF, input is mapped by the SYMBFP DSECT of the ASASYMBP mapping macro.

Because the SYMBP and SYMBFP DSECTs are much the same, except for different field names and one extra field (SymbfpWorkareaAddr) in SYMBFP, you can build the SYMBFP area in advance and then decide at runtime whether to call or LINK to ASASYMBM or ASASYMBF.

Environment

The requirements for the caller are:

Environmental factor	Requirement
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 or ASASYMBF, you must include the ASASYMBP mapping macro (see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary)).
2. Before calling ASASYMBM or ASASYMBF, 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 or ASASYMBF, 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 or ASASYMBF, the caller must examine the fullword pointed to by SYMBPTARGETLENGTH@ in the ASASYMBP mapping macro. The output itself is in the area provided by the caller, which is pointed to by SYMBPTARGET@ in ASASYMBP or ASASYMBF.

For more information about providing input to ASASYMBM or ASASYMBF 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 or ASASYMBF, the caller must ensure that the following general purpose registers (GPRs) contain the specified information:

Register
Contents

13
Address of a standard 72-byte save area in the primary address space

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

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

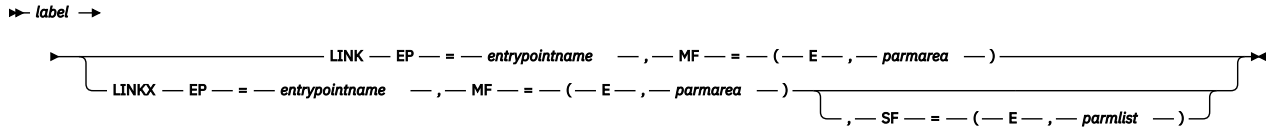
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance implications

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 or ASASYMBF service:



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 or ASASYMBF 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=*entrypointname*
Specifies the entry point name, ASASYMBM or ASASYMBF, for the service.

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

Return and reason codes

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

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The ASASYMBM or ASASYMBF request completed successfully. The system performed the requested substitution. **Action**: None required.

Hexadecimal Return Code	Meaning and Action
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 or ASASYMBF, 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 *z/OS MVS Initialization and Tuning Reference*. Ensure that the symbols in the input pattern conform to the rules for substringing.
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 or ASASYMBF, 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 or ASASYMBF

For examples of calls to ASASYMBM or ASASYMBF, see the section that describes the symbol substitution service in *z/OS MVS Programming: Assembler Services Guide*.

Chapter 6. ATTACH and ATTACHX — Create a new task

ATTACH and ATTACHX 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.

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:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN

Environmental factor	Requirement
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:

Register
Contents

0
Used as a work register by the system

1
If GPR 15 contains a return code other than X'00', zero; otherwise, the address of the task control block for the new task

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0
Used as a work register by the system

1
Zero (the ALET of the task control block address)

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 ATTACH macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ATTACH.
ATTACH	
␣	One or more blanks must follow ATTACH.
EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address, or register (2) - (12).
DE=*list entry addr*	*list entry addr*: A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,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*: A-type address

Syntax	Description
,PARAM=(*addr*),VL=1	**Note**: *addr* is one or more addresses, separated by commas. For example, PARAM=(*addr,addr,addr*)
,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*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,GSPL=*subpool list addr*	*subpool list addr*: A-type address, or register (2) - (12).
,SHSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,SHSPL=*subpool list addr*	*subpool list addr*: A-type address, or register (2) - (12).
,SZERO=YES	**Default**: SZERO=YES
,SZERO=NO	
,TASKLIB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,STAI=(*exit addr*)	*exit addr*: A-type address, or register (2) - (12).
,STAI=(*exit addr,parm addr*)	*parm addr*: A-type address, or register (2) - (12).
,ESTAI=(*exit addr*)	**Note**: AR mode callers and 31-bit callers must not use STAI.
,ESTAI=(*exit addr,parm addr)*	
,PURGE=QUIESCE	**Note**: PURGE may be specified only if STAI or ESTAI is specified.
,PURGE=NONE	**Default** for STAI: PURGE=QUIESCE
,PURGE=HALT	**Default** for ESTAI: PURGE=NONE
,ASYNCH=NO	**Default** for STAI: ASYNCH=NO
,ASYNCH=YES	**Default** for ESTAI: ASYNCH=YES
	Note: ASYNCH may be specified only if STAI or ESTAI is specified.
,TERM=NO	**Note**: TERM may be specified only if ESTAI is specified.
,TERM=YES	**Default**: TERM=NO
,ALCOPY=NO	**Default**: ALCOPY=NO
,ALCOPY=YES	

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

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 IEAAPFxx 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:

Register
Contents

0
Used as a work register by the system.

1
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.

2-12
Used as work registers by the system.

13
Address of a 144-byte 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'

Note: For assistance in converting a program to use relative branching, refer to the IEABRC and IEABRCX macros.

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

Register
Contents

0
Used as a work register by the system.

1
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.

2-12
Used as work registers by the system.

13-15
Zeros.

,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 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.

,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.

,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:

Environmental factor	Requirement
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:	24-bit when the issuer of the ATTACH macro is AMODE 24; Otherwise, 31-bit.
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:

Register
Contents

0
Used as a work register by the system.

1
Address of the task control block for the task that was terminated.

2-12
Used as work registers by the system.

13
Address of a 72-byte area provided by the system.

14
Return address (to the system).

15
Address of the exit routine.

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

Register
Contents

0
Used as a work register by the system.

1
Zero

2-12
Used as work registers by the system.

13-15
Zeros

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.

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=QUIESCE
,PURGE=NONE
,PURGE=HALT
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*.

,ASYNCH=NO
,ASYNCH=YES
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=NO
,TERM=YES
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
- z/OS UNIX System Services is canceled and the user's task is in a wait in the z/OS UNIX System Services kernel.

,ALCOPY=NO
,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.

,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:

ABEND Code	Associated Reason Code
12A	0, 4
22A	0
42A	None
52A	0, 4, 8
82A	None
92A	0, 4, 8, C, 10, 14, 18

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:

Hexadecimal Return Code	Meaning and Action
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.
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.
20	**Meaning**: Program error, due to one of the following reasons: • The current task was not subspace active and the ATTACHX macro specified ADDRENV=SUBSP. • The current task is a subspace task that is not subspace active and issued either ATTACH, or ATTACHX with ADDRENV=SAME specified or defaulted. **Action:** • If the current task was not subspace active and the ATTACHX macro specified ADDRENV=SUBSP, update your program so that it issues ATTACHX with ADDRENV=SUBSP only if it is subspace active. • If the current task is a subspace task that is not subspace active and ADDRENV=SAME was specified or defaulted, update your program so that it issues ATTACH, or ATTACHX with ADDRENV=SAME specified or defaulted, only if it is not a subspace task or is a subspace task that is not subspace active.

Hexadecimal Return Code	Meaning and Action
24	**Meaning**: Program error. ADDRENV=SAME was specified or defaulted and the issuer was a subspace task that is subspace active, but the task was processing with a different active subspace than that which was in effect when it was attached. **Action:** Update your program if it is a subspace task and subspace active so that it issues ATTACH or ATTACHX with ADDRENV=SAME only if the task was processing with the same active subspace that was in effect when it was attached.

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
PARM1          DS F
PARM2          DS F
```

```
PARM3          DS F
ATTACH_EXEC    ATTACHX SF=L
```

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ATTACHX.
ATTACHX	
␣	One or more blanks must follow ATTACHX.
EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address, or register (2) - (12).
DE=*list entry addr*	*list entry addr:* A-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,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*: A-type address
,PARAM=(*addr*),VL=1	**Note**: *addr* is one or more addresses, separated by commas. For example, PARAM=(*addr,addr,addr*)
,PLIST4=YES	**Default**: None.
,PLIST4=NO	

Syntax	Description
,PLIST8=YES	**Default**: None.
,PLIST8=NO	
,PLISTARALETS=SYSTEM	**Default**: ,PLISTARALETS=SYSTEM
,PLISTARALETS=NO	**Note**: ,PLISTARALETS is valid only with ATTACHX.
,PLIST8ARALETS=NO	**Default**: PLIST8ARALETS=NO
,PLIST8ARALETS=YES	
,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*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,GSPL=*subpool list addr*	*subpool list addr*: A-type address, or register (2) - (12).
,SHSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,SHSPL=*subpool list addr*	*subpool list addr*: A-type address, or register (2) - (12).
,SZERO=YES	**Default**: SZERO=YES
,SZERO=NO	
,TASKLIB=*dcb addr*	*dcb addr*: A-type address, or register (2) - (12).
,STAI=(*exit addr)*	*exit addr*: A-type address, or register (2) - (12).
,STAI=(*exit addr,parm addr)*	*parm addr*: A-type address, or register (2) - (12).
,ESTAI=(*exit addr*)	**Note**: AR mode callers and 31-bit callers must not use STA.
,ESTAI=(*exit addr,parm addr*)	
,SDWALOC31=NO	**Note:** SDWALOC31 may be specified only when ESTAI is specified.
,SDWALOC31=YES	**Default:** SDWALOC31=NO
,PURGE=QUIESCE	**Note**: Specify PURGE only if you specify ESTAI.
,PURGE=NONE	**Default** for ESTAI: PURGE=NONE
,PURGE=HALT	

Syntax	Description
,ASYNCH=NO	**Note**: Specify ASYNCH only if you specify ESTAI.
,ASYNCH=YES	**Default** for ESTAI: ASYNCH=YES
,TERM=NO	**Note**: Specify TERM only if you specify ESTAI.
,TERM=YES	**Default**: TERM=NO
,ALCOPY=NO	**Default**: ALCOPY=NO
,ALCOPY=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,KEY=PROP	**Default:** KEY=PROP
,KEY=NINE	
,PKM=SYSTEM_RULES	**Default:** PKM=SYSTEM_RULES
,PKM=REPLACE	
,ADDRENV=SAME ,ADDRENV=SUBSP	**Default**: ADDRENV=SAME

Parameters

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

,PARAM=(*addr*)
,PARAM=(*addr*),VL=1
Specifies an address or addresses to be passed to the attached task. ATTACHX expands each address inline to a fullword boundary and builds a 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 without specifying PLISTARALETS=NO; 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 4 bytes per entry with PLISTARALETS=NO, 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 PLISTARALETS and PLIST8ARALETS keyword below for more information about

ALETs and 8-bytes-per-entry parameter lists. See User parameters 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.

,PLISTARALETS=SYSTEM
,PLISTARALETS=NO

If the invoker is in AR mode, indicates whether the parameter list is also to contain the ALETs associated with the addresses. If the invoker is not in AR mode, this parameter is ignored.

,PLISTARALETS=SYSTEM

Indicates to follow the default system rules that for an AR mode invoker:

- For AMODE 24/31, the parameter list is also to contain the ALETs.
- For AMODE 64 with PLIST8ARALETS=YES, the parameter list is also to contain the ALETs.
- For other cases, the parameter list is not to contain the ALETs.

,PLISTARALETS=NO

Indicates that the parameter list is not also to contain the ALETs. Do not specify this parameter with PLIST8ARALETS=YES.

,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.

,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: GSPL, GSPV, SHSPL, and SHSPV. 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)
- the CEL anchor pointer is above 16M. For a task that is not KEY=NINE, the CEL anchor pointer 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.

,ADDRENV=SAME
,ADDRENV=SUBSP
Identifies processing related to the subspace environment for the new task. In general, the program is responsible for keeping track of whether it is a subspace task or whether it is subspace active.

A subspace task is a task that was attached either by ATTACHX with ADDRENV=SUBSP or by a task that itself was a subspace task that was subspace active at the time of the ATTACH or ATTACHX.

Note: It is up to the program that issues BSG to keep track of whether it is subspace active.

,ADDRENV=SAME
If the current task is a subspace task and is active to the same active subspace that was in effect when the current task was attached, make the new task a subspace task that is active to that subspace. If the current task is not a subspace task, take no action. Do not use this option if the current task is a subspace task that either is not subspace active or is subspace active but for a different subspace than was in effect when the current task was attached. The Access List Entry for the home ALET (ALET 2) of the new task will be set to address the subspace instead of the home base space.

,ADDRENV=SUBSP
If the current task is a subspace task and is subspace active, make the new task a subspace task and active to that subspace. Do not specify this option if the current task is not subspace active.

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. These parameter lists can be referred to in the execute form.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ATTACH or ATTACHX.
ATTACH ATTACHX	
␣	One or more blanks must follow ATTACH or ATTACHX.
EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: A-type address.
DE=*list entry addr*	*list entry addr*: A-type address.
,DCB=*dcb addr*	*dcb addr*: A-type address.

Syntax	Description
,LPMOD=*limit prior nmbr*	*limit prior nmbr*: Symbol or decimal digit.
,DPMOD=*disp prior nmbr*	*disp prior nmbr*: Symbol or decimal digit.
,PLISTARALETS=SYSTEM	**Default**: ,PLISTARALETS=SYSTEM
,PLISTARALETS=NO	**Note**: ,PLISTARALETS is valid only with ATTACHX.
,PLIST8ARALETS=NO	**Default**: PLIST8ARALETS=NO
,PLIST8ARALETS=YES	**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.
,GSPV=*subpool nmbr*	*subpool nmbr*: Symbol or decimal digit.
,GSPL=*subpool list addr*	*subpool list addr*: A-type address.
,SHSPV=*subpool nmbr*	*subpool nmbr*: Symbol or decimal digit.
,SHSPL=*subpool list addr*	*subpool list addr*: A-type address.
,SZERO=YES	**Default**: SZERO=YES
,SZERO=NO	
,TASKLIB=*dcb addr*	*dcb addr*: A-type address.
,STAI=(*exit addr*)	*exit addr*: A-type address.
,STAI=(*exit addr,parm addr*)	*parm addr*: A-type address.
,ESTAI=(*exit addr*)	**Note**: AR mode callers and 31-bit callers must not use STA.
,ESTAI=(*exit addr,parm addr*)	
,SDWALOC31=NO	**Note:** SDWALOC31 is valid only with ATTACHX AND when ESTAI is specified.
,SDWALOC31=YES	**Default:** SDWALOC31=NO
,PURGE=QUIESCE	**Note**: PURGE may be specified only if STAI or ESTAI is specified.
,PURGE=NONE	**Default** for STAI: PURGE=QUIESCE
,PURGE=HALT	**Default** for ESTAI: PURGE=NONE

Syntax	Description
,ASYNCH=NO ,ASYNCH=YES	**Note**: ASYNCH may be specified only if STAI or ESTAI is specified. **Default** for STAI: ASYNCH=NO **Default** for ESTAI: ASYNCH=YES
,TERM=NO	**Note**: TERM may be specified only if ESTAI is specified.
,TERM=YES	**Default**: TERM=NO
,ALCOPY=NO ,ALCOPY=YES	**Default**: ALCOPY=NO
,RELATED=*value*	*value*: Any valid macro keyword specification.
,KEY=PROP	**Default:** KEY=PROP
,KEY=NINE	**Note:**KEY=NINE is valid only when using ATTACHX.
,PKM=SYSTEM_RULES	**Default:** PKM=SYSTEM_RULES
,PKM=REPLACE	**Note:**PKM is valid only when using ATTACHX.
,ADDRENV=SAME ,ADDRENV=SUBSP	**Default**: ADDRENV=SAME
,SF=L	

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:

,SF=L
Specifies the list form of 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ATTACH or ATTACHX.
ATTACH ATTACHX	
␣	One or more blanks must follow ATTACH or ATTACHX.
EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: RX-type address, or register (2) - (12).
DE=*list entry addr*	*list entry addr*: RX-type address, or register (2) - (12).
,DCB=*dcb addr*	*dcb addr:*: RX-type address, or register (2) - (12).
,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.

Syntax	Description
,PLIST8=NO	**Default**: None.
,PLISTARALETS=SYSTEM	**Default**: ,PLISTARALETS=SYSTEM
,PLISTARALETS=NO	**Note**: ,PLISTARALETS is valid only with ATTACHX.
,PLIST8ARALETS=NO	**Default**: PLIST8ARALETS=NO
,PLIST8ARALETS=YES	**Note**: PLIST8ARALETS is valid only with ATTACHX.
,ECB=*ecb addr*	*ecb addr*: RX-type address, or register (2) - (12).
,ETXR=*exit rtn addr*	*exit rtn addr*: RX-type address, or register (2) - (12).
,GSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,GSPL=*subpool list addr*	*subpool list addr*: RX-type address, or register (2) - (12).
,SHSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,SHSPL=*subpool list addr*	*subpool list addr*: RX-type address, or register (2) - (12).
,SZERO=YES ,SZERO=NO	
,TASKLIB=*dcb addr*	*dcb addr*: RX-type address, or register (2) - (12).
,STAI=(*exit addr*)	*exit addr*: RX-type address, or register (2) - (12).
,STAI=(*exit addr,parm addr*)	*parm addr*: RX-type address, or register (2) - (12).
,ESTAI=(*exit addr*)	**Note**: AR mode callers and 31-bit callers must not use STA.
,ESTAI=(*exit addr,parm addr*)	
SDWALOC31=NO	**Note:** SDWALOC31 is valid only when using ATTACHX AND when ESTAI is specified.
SDWALOC31=YES	**Default:** SDWALOC31=NO
,PURGE=QUIESCE	**Note**: PURGE may be specified only if STAI or ESTAI is specified.
,PURGE=NONE	
,PURGE=HALT	

Syntax	Description
,ASYNCH=NO ,ASYNCH=YES	**Note**: ASYNCH may be specified only if STAI or ESTAI is specified.
,TERM=NO	**Note**: TERM may be specified only if ESTAI is specified.
,TERM=YES	
,ALCOPY=NO	**Default**: ALCOPY=NO
,ALCOPY=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,KEY=PROP	**Default:** KEY=PROP
,KEY=NINE	**Note:**KEY=NINE is valid only when using ATTACHX
,PKM=SYSTEM_RULES	**Default:** PKM=SYSTEM_RULES
,PKM=REPLACE	**Note:**PKM is valid only when using ATTACHX.
,MF=(E,*prob addr*)	*prob addr*: RX-type address, or register (1) or (2) - (12).
,SF=(E,*ctrl addr*) ,MF=(E,*prob addr*),SF=(E,*ctrl addr*)	*ctrl addr*: RX-type address, or register (2) - (12) or (15).
,ADDRENV=SAME ,ADDRENV=SUBSP	**Default**: ADDRENV=SAME

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.

Note:

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.

Chapter 7. BCFXCRYP — Interface to invoke the protected key encryption service

Description

The BCFXCRYP macro invokes BCFCRYPT to perform an encryption operation or a related function. It uses a CRYPI and CRYPV for most of the functions. These control blocks are documented under BCFZCRYP in *z/OS MVS Data Areas Volume 1 (ABE - IAR)* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or Access Register ASC mode If in access register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt Status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked
Control parameters:	Not applicable

Programming requirements

Before invoking BCFXCRYP, you must obtain storage for the CRYPI, CRYPV, and CRYPWA. You must include the mapping macro BCFZCRYP. *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for more information on BCFZCRYP.

Restrictions

None.

Input register information

Before issuing the BCFXCRYP 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 general purpose registers (GPRs) contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a 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 BCFXCRYP macro is written as follows:

Syntax	Description
xlabel	*name*: Symbol. Begin *xlabel* in column 1.
ƀ	One or more blanks must precede BCFXCRYP.
BCFXCRYP	
ƀ	One or more blanks must follow BCFXCRYP.
workarea	*workarea*: Name (RS-type) or address in register (2) - (12).
,cryp_block	*,cryp_block*: Name (RS-type) or address in register (2) - (12).

Syntax	Description
,request	*,request*: Optional One of the following literals: INIT_CRYPI FLUSH_STATS

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

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

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

Syntax	Description
xlabel	*name*: Symbol. Begin *xlabel* in column 1.
ƀ	One or more blanks must precede BCFXCRYP.
BCFXCRYP	
ƀ	One or more blanks must follow BCFXCRYP.
workarea	*workarea*: Name (RS-type) or address in register (2) - (12).
,cryp_block	*,cryp_block*: Name (RS-type) or address in register (2) - (12).

Syntax	Description
,request	*,request*: Optional One of the following literals: INIT_CRYPI FLUSH_STATS
MF=(E, *plabel*)	*plabel* is the *plabel* specified on the list form.

Parameters

The parameters are explained as follows:

workarea
The 512-byte work area for use by the BCFCRYPT service. The work area does not need to be initialized prior to the first use, nor does it need to be cleared prior to subsequent uses. It does need to be cleared prior to free.

,cryp_block
Either the CRYPV or CRYPI parm block.

,request
Indicates the type of operation requested.

INIT_CRYPI
This invocation is to set up the initial query. The cryp_block passed must be a CRYPI, and there must be only one task accessing the CRYPI.

FLUSH_STATS
This invocation is to force recording of any crypto statistics data before freeing the CRYPI. The cryp_block passed must be a CRYPI, and there must be only one task accessing the CRYPI.

nil (neither INIT_CRYPI nor FLUSH_STATS specified)
This invocation is perform the operation requested in the CRYPV. The *cryp_block* passed must be a CRYPV.

,MF=L
Specifies the list form of BCFXCRYP.

MF=(E,*plabel*)
Specifies the execute form of the BCFXCRYP macro using the previously defined parameter area.

Return and reason codes

When BCFXCRYP completes, register 15 contains a return code and register 0 contains a reason code. Refer to macro BCFZCRYP for the equate symbols and complete descriptions for the return and reason codes.

Example

Full samples for AMODE(31) and AMODE(64) are available in SAMPLIB as BCFC31AS and BCFC64AS, respectively.

Chapter 8. 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* for more information on using the BLDMPB macro.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN 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* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) 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:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BLDMPB.
BLDMPB	
ƀ	One or more blanks must follow BLDMPB.
MPBPTR=*mpb addr*	*mpb addr:* RX-type address or register (2) - (12).
,MPBLEN=*mpb length addr*	*mpb length addr*: RX-type address or register (2) - (12).
,MSGID=*msg id addr*	*msg id addr:* RX-type address or register (2) - (12).
,MSGIDLEN=*msg id length*	*msg id length addr*: RX-type address or register (2) - (12).
addr	
,MSGFMTNM=*format num*	*format num addr*: RX-type address or register (2) - (12).

Syntax	Description
addr	
,MSGLNNM=*line num addr*	*line num addr*: RX-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

MPBPTR=*mpb addr*
Specifies the address or a register containing the address of the 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 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:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00	00	Successful processing.
0C	33	The MPB is too small.
0C	34	The value for MSGIDLEN is zero or negative.

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
*
*************************************************************************
*        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,SUBOOFST=VARS,               C
               TOKEN=TOKN,TOKLEN=TOKL,TOKTYPE=TOKT,                   C
               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
*************************************************************************
MPBL     DC    A(MPBLEN)           ADDRESS OF MPB LENGTH
MSGID    DC    CL10'MSGID2'        MSG ID OF MESSAGE REPRESENTED BY MPB
MIDLEN   DC    A(MIDL)             ADDRESS OF MSG ID LENGTH
TOKN     DC    CL3'DAY'            TOKEN NAME
TOKL     DC    F'3'                LENGTH OF TOKEN NAME
TOKT     DC    CL1'3'              TOKEN TYPE (DAY)
SDATA    DC    CL1'3'              SUBSTITUTION DATA (3RD DAY OF WEEK)
SDATAL   DC    A(SDL)              ADDRESS OF SUBSTITUTION DATA LENGTH
SAVE     DC    18F'0'              SAVE AREA
SP230    EQU   230                 SUBPOOL SPECIFICATION FOR GETMAIN
STORLEN  EQU   256                 LENGTH OF GETMAINED STORAGE
SDL      EQU   6                   SUBSTITUTION DATA LENGTH
MIDL     EQU   6                   MSG ID LENGTH
MPBLEN   EQU   (MPBVDAT-MPB)+(MPBMID-MPBMSG)+(MPBSUB-MPBSB)+MIDL+SDL C
                                   TOTAL MPB LENGTH
R1       EQU   1                   REGISTER 1
R2       EQU   2                   REGISTER 2
R4       EQU   4                   REGISTER 3
*************************************************************************
         DSECT
         CNLMMPB
VARS     DSECT
VARSAREA DS    CL24
VARSLEN  EQU   *-VARS
         END BLDMPBA
```

Chapter 9. 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.

See *z/OS MVS IPCS Customization* for information about IPCS exit services; see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSABDPL data area.

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:

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

Syntax	Description
ƀ	One or more blanks must precede BLSABDPL.
BLSABDPL	
ƀ	One or more blanks must follow BLSABDPL.
,AMDCPST=YES	
,AMDCPST=NO	**Default**: AMDCPST=NO
,AMDEXIT=YES	**Default**: AMDEXIT=YES
,AMDEXIT=NO	
,AMDOSEL=YES	**Default**: AMDOSEL=YES
,AMDOSEL=NO	
,AMDPACC=YES	**Default**: AMDPACC=YES
,AMDPACC=NO	
,AMDPECT=YES	**Default**: AMDPECT=YES
,AMDPECT=NO	
,AMDPFMT=YES	**Default**: AMDPFMT=YES
,AMDPFMT=NO	
,AMDPSEL=YES	**Default**: AMDPSEL=YES
,AMDPSEL=NO	
,DSECT=YES	**Default**: DSECT=YES
,DSECT=NO	

Parameters

The parameters are explained as follows:

,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.

,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.

,AMDOSEL=YES
,AMDOSEL=NO
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.

,AMDPACC=YES
,AMDPACC=NO
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.

,AMDPECT=YES
,AMDPECT=NO
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.

,AMDPFMT=YES
,AMDPFMT=NO
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.

,AMDPSEL=YES
,AMDPSEL=NO
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.

,DSECT=YES
,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.

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

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.

```
BLSABDPL DSECT=NO,AMDEXIT=NO,AMDOSEL=NO,AMDPACC=NO,
     AMDPFMT=NO,AMDPECT=NO,AMDPSEL=YES
```

Chapter 10. 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.

See *z/OS MVS IPCS Customization* for information about the CBSTAT exit service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSACBSP data area.

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:

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

Syntax	Description
DSECT=YES	**Default**: DSECT=YES
DSECT=NO	
ABITS=31	**Default**: ABITS=31
ABITS=64	

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 11. 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* for information about the add symptom service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) 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:

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

Syntax	Description
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 12. 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.

See *z/OS MVS IPCS Customization* for information about the CQE create service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSAPCQE data area.

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BLSAPCQE.
BLSAPCQE	
ƀ	One or more blanks must follow BLSAPCQE.
DSECT=YES	**Default**: DSECT=YES

Syntax	Description
DSECT=NO	

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
DSECT=NO
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.

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 13. 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.

See *z/OS MVS IPCS Customization* for information about model processor formatting exit routines. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSQFXL data area.

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BLSQFXL.
BLSQFXL	BLSQFXL must begin in column 1.
ƀ	One or more blanks must follow BLSQFXL.
DSECT=YES	**Default**: DSECT=YES

Syntax	Description
DSECT=NO	

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 14. 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* 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:

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

Syntax	Description
␣	One or more blanks must precede BLSQMDEF.
BLSQMDEF	
␣	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

Syntax	Description
,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.

,CBLEN=*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 ACROLBL or ACROFF parameter should also be specified to define the offset of the acronym field within the control block. When neither the ACROLBL 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.

,ACROLBL=*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 ACROLBL 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.

,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.

The PREFIX parameter on the BLSQMDEF and BLSQMFLD macros behave differently. The PREFIX parameter on the BLSQMFLD macro allows you to specify a prefix value of 8 or more to map fields with names that are more than 8 characters long. The PREFIX parameter on the BLSQMDEF macro does not allow more than 8 characters for compatibility reasons. Note that specifying a PREFIX value of 8 on the BLSQMDEF macro does not mean that characters 9 - 16 of a 16-character variable will be displayed; rather, a value of 8 indicates the no-label mode, as described earlier. If you want a prefix of 8 or more, you must specify a PREFIX value less than 8 on the BLSQMDEF invocation and then specify your desired prefix on a BLSQMFLD invocation.

,OFFSETS=PRINT
,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.

,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.

,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.

Note: When *value* is 18, the output is condensed.

,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.

,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 15. 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:

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

Syntax	Description
BLSQMFLD	
␣	One or more blanks must follow BLSQMFLD.
NAME=*name*	
NAME=*	*name*: Symbol.
,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)*	*(list)*: Integers between 1 and 16, inclusive.
,VIEW=*value*	*value*: Decimal constant, hexadecimal constant, or absolute value.
	Default: VIEW=X‘0200’.
,ARRAY=*constants*	*constants*: ((DL1,DU1),(DL2,DU2))
,ARRAY=*value*	*DL1,DU1,DL2,DU2:* Decimal constants, hexadecimal constants, or absolute values.
,ARRAY=*	*value:* Decimal constant, hexadecimal constant, or absolute value.
,ARRAY=END	**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	
,DTYPE=ASCII	
,DECODE	

Syntax	Description
,INVERT	
,ATTACH	
,IMBED	
,STACK	
,CALLCBF	
,NEWLINE	
,NOLABEL	
,CALLRTN	
,LABEL=*label*	*label*: Symbol.
,LAST8	
,PREFIX=*value*	*value*: Integer 0 or greater.
	Note: When omitted, *value* specified in the last preceding BLSQMDEF or BLSQMFLD macro is used.
,NOSPLIT	
,NUMDEC	**Default**: Hexadecimal.
,NOCOLNM	**Default**: Number the columns.
,NOROWNM	
,STRTCOL=*value*	*value*: Decimal constant, hexadecimal constant, or absolute value.
	Default: Value specified by IPCS or SNAP.
,COLNUM=*value*	*value*: Decimal constant, hexadecimal constant, or absolute value.

Syntax	Description
	Default: A value is calculated.
,COLSEP=*value*	*value*: Decimal constant, hexadecimal constant, or absolute value.
	Default: A value is calculated.
,ITEMSEP=*value*	*value*: Decimal constant, hexadecimal constant, or absolute value.
	Default: A value is calculated.
,ORDER=(1,2)	**Default**: ORDER=(1,2)
,ORDER=(2,1)	
,HEXONLY	
,MODELNAME(modelname)	
,MSGID(msgid)	
,SRCNDX	**Default**: 0

Parameters

The parameters are explained as follows:

NAME=*name*
NAME=*

Specifies the name of the control block field described by the BLSQMFLD macro. When BASELBL is specified on the BLSQMDEF macro, the NAME parameter is used with the BASELBL parameter 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.

When the LABEL parameter is specified, NAME is only used to calculate length and offset, and the LABEL text appears next to the data. When LABEL is not specified, NAME is used to produce the field label.

,LABEL=*label*

Specifies up to 8 characters to be displayed as the field label. These characters will appear before the contents of the requested field. If not specified, the NAME field will be used to produce the field label.

,LAST8

The last 8 characters specified in the NAME field are used as the field label. Any previous PREFIX specification is ignored. LAST8 is only valid when NAME is more than 8 characters long. LAST8 is not valid when SHDR, LABEL or PREFIX are specified.

,SHDR=*addr*
Specifies the address of a character string used as a subheading in the control 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 parameter on this macro and the BASELBL parameter 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 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'.).

The following 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.

Hexadecimal code	User-defined fields to be displayed
X'8000'.	Keyfield
X'4000'.	Summary field
X'2000'.	Register save area
X'1000'.	Linkage field

Hexadecimal code	User-defined fields to be displayed
X'0800'.	Error fields
X'0400'.	Hexadecimal dump
X'0200'.	Non-reserved field
X'0100'.	Reserved fields
X'0080'.	Static array or decode flag fields
X'0040'.	Dynamic array
X'0020'.	Input field
X'0010'.	Output field

When this parameter is not specified, the default value of VIEW=X'0200'. is used. See *z/OS MVS IPCS User's Guide* and *z/OS MVS IPCS Commands* for more information about ADPLPFMT.

,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=*value* is coded, a one dimensional array (list) is specified. *Value* defines the number of array entries contained in the control block.

When ARRAY=* 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.

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

Specifies the type of data contained in the area to be displayed. DTYPE=HEX indicates that the area to be displayed contains four-bit hexadecimal digits. DTYPE=EBCDIC indicates that the area to be dumped contains eight-bit EBCDIC characters. DTYPE=ASCII indicates that the area to be dumped contains ASCII characters.

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.

DTYPE=QANY specifies that the next entry in the model is a subheader entry, with a view field of all zero. The value of the field is only treated as EBCDIC when it is the same as one of the values specified in the text of the subheader. Otherwise, the field is displayed in hexadecimal format. 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.

,DECODE

Specifies that the model entry describes one of these decoding operations:

- Flag field decoding
- Format imbedded block
- Format attached block
- Format stacked block

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.

DECODE is not supported within an array.

,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 NAME parameter 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. LABEL will be ignored when NOLABEL is specified.

,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. 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 LABEL or LAST8 is specified with PREFIX, IPCS adjusts the global prefix value but the field label produced is not determined by the PREFIX value. LABEL causes the first 8 characters given on the LABEL parameter to be shown as the field label. LAST8 causes the last 8 characters of the NAME parameter to be shown as the field label. PREFIX specification affects all future BLSQMFLD invocations unless PREFIX is specified again.

,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.

Examples

The following examples demonstrate the use of the BLSQMFLD macro.

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 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'0200'
         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=FRRSKM,OFF=X'00A0',LEN=2,VIEW=X'0200',NEWLINE
         BLSQMFLD NAME=FRRSSAS,OFF=X'00A2',LEN=2,VIEW=X'0200'
         BLSQMFLD NAME=FRRSAX,OFF=X'00A4',LEN=2,VIEW=X'0200'
         BLSQMFLD NAME=FRRSPAS,OFF=X'00A6',LEN=2,VIEW=X'0200',ARRAY=END
         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',       X
               ARRAY=END
         BLSQMDEF END
BLANK    BLSQSHDR ' '
ENTEXT   BLSQSHDR 'FRR ENTRY EXTENSIONS'
ENTS     BLSQSHDR 'FRR ENTRIES'
RTM1WA   BLSQSHDR 'RTM1 WORK AREA FOLLOWS FRR ENTRIES'
         END
```

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 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=SCBFLGS1,OFF=X'0008',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBPARMA,OFF=X'0009',LEN=3,VIEW=X'0200'
         BLSQMFLD NAME=SCBFLGS2,OFF=X'000C',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBOWNRA,OFF=X'000D',LEN=3,VIEW=X'0200'
         BLSQMFLD NAME=SCBFLGS3,OFF=X'0010',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBPKEY,OFF=X'0011',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBID,OFF=X'0012',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBRSVRE,OFF=X'0013',LEN=1,VIEW=X'0200'
         BLSQMFLD NAME=SCBXPTR,OFF=X'0014',LEN=4,VIEW=X'0200'
         BLSQMFLD NAME=*,OFF=X'0000',LEN=X'0018',VIEW=X'0400',NOLABEL
         BLSQMDEF END

         END
```

Example 3

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

```
MYBLK    DSECT ,                  My simplest control block ever
MYBLKABC DC    C'ABC'             Identifier
MYBLKDEF DC    X'00'              Flags
MYBLKD80 EQU   X'80'              1st flag bit
MYBLKD40 EQU   X'40'              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.

```
         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.

```
         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
         END   CBMODEL1           End definition of formatting model
```

Example 4

Assume that the data is stored in this sequence:

```
00010001
00010002
00010003
00010004
00020001
00020002
00020003
00020004
00030001
00030002
00030003
00030004
   .
   .
   .
00090001
00090002
00090003
00090004
00100001
00100002
00100003
00100004
```

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
00090002
00090003
00090004
00100001
00100002
00100003
00100004
```

And you want the data to be formatted this way:

```
    ---05--- ---06--- ---07--- ---08--- ---09---
    ARRENTRY ARRENTRY ARRENTRY ARRENTRY ARRENTRY
    -------- -------- -------- -------- --------
000 00010001 00020001 00030001 00040001 00050001
001 00010002 00020002 00030002 00040002 00050002
002 00010003 00020003 00030003 00040003 00050003
003 00010004 00020004 00030004 00040004 00050004
    ---0A--- ---0B--- ---0C--- ---0D--- ---0E---
    ARRENTRY ARRENTRY ARRENTRY ARRENTRY ARRENTRY
    -------- -------- -------- -------- --------
000 00060001 00070001 00080001 00090001 00100001
001 00060002 00070002 00080002 00090002 00100002
002 00060003 00070003 00080003 00090003 00100003
003 00060004 00070004 00080004 00090004 00100004
```

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:

```
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'
```

Example 6

When presented with a field with a long name, you can use the LABEL parameter to assign the characters you want to appear next to the field contents.

For example, consider the following control block:

```
MYBLK    DSECT ,                  My simple control block
MYBLKABC DC    C'ABC'             Identifier
MYBLKDEF DC    X'00'              Flags
MYBLKD80 EQU   X'80'              1st flag bit
MYBLKD40 EQU   X'40'              2nd flag bit
MYBLKGHI DC    V(MYENTRY)         Address of my program
MYBLK_LONG_LABEL_FIELD DC  C'xxxxxxxx' Field with a long label
MYBLKEND EQU   *
```

You can define a model as follows:

```
         TITLE 'CBMODEL3- Long label tests'
         ENTRY CBMODEL3
CBMODEL3 BLSQMDEF BASELBL=MYBLK,CBLEN=MYBLKLEN,PREFIX=5,HEADER=MYBLOCK
         BLSQMFLD NAME=MYBLKABC,DTYPE=EBCDIC
         BLSQMFLD NAME=MYBLKDEF
         BLSQMFLD NAME=MYBLKGHI
         BLSQMFLD NAME=MYBLK_LONG_LABEL_FIELD
         BLSQMFLD NAME=MYBLK_LONG_LABEL_FIELD,LABEL=LONGLBLF
         BLSQMFLD NAME=MYBLK_LONG_LABEL_FIELD,LAST8
         BLSQMDEF END
```

The result when used in IPCS as a control block formatter will look like this:

```
MYBLOCK: 00000000
   +0000  ABC...... ABC       DEF...... 80        GHI...... 00513000  LONGLBLF. 00000000
00000000
   +0008  _LONG_LA. 00000000  00000000  LONGLBLF. 00000000  00000000
   +0008  EL_FIELD. 00000000  00000000
```

Chapter 16. 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* 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.

Syntax

This is the standard form of the BLSQSHDR macro:

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

L(*x*)
specifies the length of the subheader. Specify the length only when it is different from the length of the enclosed text string.

Examples

```
SHDR01  BLSQSHDR 'This is a subheader'
SHDR02  BLSQSHDR L(6)' '
```

Chapter 17. 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.

See *z/OS MVS IPCS Customization* for information about formatting dump data. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSRDRPX data area.

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:

Syntax	Description
name	*name*: Symbol. *name* must begin in column 1 and it cannot exceed four characters in length.
ƀ	One or more blanks must precede BLSRDRPX.
BLSRDRPX	
ƀ	One or more blanks must follow BLSRDRPX.
ABITS=31	**Default**: ABITS=64
ABITS=64	

Syntax	Description
DSECT=YES	**Default**: DSECT=YES
DSECT=NO	

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 18. 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 the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BLSRESSY.
BLSRESSY	
ƀ	One or more blanks must follow BLSRESSY.
DSECT=YES DSECT=NO	**Default**: DSECT=YES
ABITS=31	**Default**: ABITS=31

Syntax	Description
ABITS=64	

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 19. 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)

See *z/OS MVS IPCS Customization* for information about the name service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSRNAMP data area.

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:

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

Syntax	Description
ƀ	One or more blanks must follow BLSRNAMP.
DSECT=YES	**Default**: DSECT=YES
DSECT=NO	

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 20. BLSRPRD — Map dump record

Description

The BLSRPRD macro creates a map of the dump record.

See *z/OS MVS IPCS Customization* for information about formatting dump data. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSRPRD data area.

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:

Syntax	Description
name	*name*: Symbol. *name* must begin in column 1 and it cannot exceed four characters in length.
ƀ	One or more blanks must precede BLSRPRD.
BLSRPRD	
ƀ	One or more blanks must follow BLSRPRD.
DSECT=YES	**Default**: DSECT=YES
	DSECT=NO

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 21. 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.

See *z/OS MVS IPCS Customization* for information about the WHERE service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSRPWHS data area.

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:

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

Syntax	Description
DSECT=YES	**Default**: DSECT=YES
DSECT=NO	
ABITS=31	**Default**: ABITS=31
ABITS=64	

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
DSECT=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
ABITS=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 22. 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 the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BLSRSASY.
BLSRSASY	
ƀ	One or more blanks must follow BLSRSASY.
DSECT=YES	**Default**: DSECT=YES
DSECT=NO	
ABITS=31	**Default**: ABITS=31

Syntax	Description
ABITS=64	

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

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

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 23. 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.

See *z/OS MVS IPCS Customization* for information about the storage map service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for a mapping of the BLSRXMSP data area.

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:

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

Syntax	Description
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 24. 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* for information about the symbol service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BLSRXSSP.
BLSRXSSP	
ƀ	One or more blanks must follow BLSRXSSP.
DSECT=YES	**Default**: DSECT=YES

Syntax	Description
DSECT=NO	

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 25. 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.

See *z/OS MVS IPCS Customization* for information about the expanded print service. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the mapping of the BLSUPPR2 data area.

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BLSUPPR2.
BLSUPPR2	
ƀ	One or more blanks must follow BLSUPPR2.

Syntax	Description
DSECT=YES	**Default**: DSECT=YES
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 26. CALL — Pass control to a control section

CALL 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* and *z/OS MVS Program Management: Advanced Facilities* 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:

Environmental factor	Requirement
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:

- 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:

Register
Contents

1
Address of the parameter list, if present

14
Return address

15
Entry address of the called program

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede CALL.
CALL	
␣	One or more blanks must follow CALL.
entry name	*entry name*: Symbol or register (2) - (12), (15).
,(*addr*)	*addr*: A-type address, or register (2) - (12).
,(*addr*),VL	**Note**: *addr* is one or more addresses, separated by commas. For example, (*addr,addr,addr*)
,PLIST4=YES	**Default**: None.
,PLIST4=NO	
,PLIST8=YES	**Default**: None.
,PLIST8=NO	
,PLIST8ARALETS=NO	**Default**: PLIST8ARALETS=NO

Syntax	Description
,PLIST8ARALETS=YES	
,ID=*id nmbr*	*id nmbr*: Symbol or decimal digit, with a maximum value of 4095.
,LINKINST=*instruction*	**Default**: LINKINST=BALR

Parameters

The parameters are explained as follows:

entry name
Specifies the entry name to be given control. When a register is specified via (n), the contents of that register are put into register 15 prior to the linkage. When (15) is specified, the system assumes that the user has set register 15 so does not do so again.

,(*addr*)
,(*addr*),VL
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 without specifying PLISTARALETS=NO; 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 4 bytes per entry with PLISTARALETS=NO, 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 PLISTARALETS and PLIST8ARALETS keyword below for more information about ALETs and 8-bytes-per-entry parameter lists. See User parameters for an example of passing a parameter list in AR mode.

When using a 4-bytes-per-entry parameter list, specify VL when you pass a variable number of parameters. VL 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 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.

,PLISTARALETS=SYSTEM
,PLISTARALETS=NO
If the invoker is in AR mode, indicates whether the parameter list is also to contain the ALETs associated with the addresses. If the invoker is not in AR mode, this parameter is ignored.

,PLISTARALETS=SYSTEM
Indicates to follow the default system rules that for an AR mode invoker:

- For AMODE 24/31, the parameter list is also to contain the ALETs.
- For AMODE 64 with PLIST8ARALETS=YES, the parameter list is also to contain the ALETs.
- For other cases, the parameter list is not to contain the ALETs.

,PLISTARALETS=NO
Indicates that the parameter list is not also to contain the ALETs. Do not specify this parameter with PLIST8ARALETS=YES.

,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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede CALL.
CALL	
␣	One or more blanks must follow CALL.
,(*addr*)	*addr*: A-type address.
,(*addr*),VL	**Note**: *addr* is one or more addresses, separated by commas. For example, (*addr,addr,addr*)
,PLISTARALETS=SYSTEM	**Default**: ,PLISTARALETS=SYSTEM
,PLISTARALETS=NO	**Note**: ,PLISTARALETS is valid only with ATTACHX.
,PLIST8ARALETS=NO	**Default**: PLIST8ARALETS=NO
,PLIST8ARALETS=YES	
,MF=L	

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede CALL.
CALL	
␣	One or more blanks must follow CALL.
entry name	*entry name*: Symbol or register (2) - (12), (15).
,(*addr*)	*addr*: RX-type address, or register (2) - (12).
,(*addr*),VL	**Note**: *addr* is one or more addresses, separated by commas. For example, (*addr,addr,addr*)
,PLISTARALETS=SYSTEM	**Default**: ,PLISTARALETS=SYSTEM
,PLISTARALETS=NO	**Note**: ,PLISTARALETS is valid only with ATTACHX.
,PLIST8ARALETS=NO	**Default**: PLIST8ARALETS=NO
,PLIST8ARALETS=YES	
,ID=*id nmbr*	*id nmbr*: Symbol or decimal digit, with a maximum value of 4095.
,LINKINST=*instruction*	**Default**: LINKINST=BALR
,MF=(E,*prob addr*)	*prob addr*: RX-type address, or register (1) or (2) - (12).

Parameters

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

,MF=(E,*prob 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 27. 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:

Environmental factor	Requirement
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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The CHAP macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CHAP.
CHAP	
ƀ	One or more blanks must follow CHAP.
priority value	*priority value*: Symbol, decimal digit, or register (0) or (2) - (12).
,*'S'*	*tcb addr*: RX-type address, or register (1) or (2) - (12).
,*tcb addr*	**Default**: 'S'
,RELATED=*value*	*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'
,tcb addr
Specifies the address of a fullword on a fullword boundary containing the address of a task control block (TCB) for a subtask of the active task. If "S" is coded or assumed, the dispatching priority of the active task is updated.

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.

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
```

Chapter 28. 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:
  - HMCS is a valid subtype for a console type of MCS.
  - SYSCON is a valid subtype for a console type of EMCS.
  - Internal, Instream, Unknown, and JES3 are valid subtypes 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:

Environmental factor	Requirement
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, CONSOLETYPE, 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:

- 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:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CnzConv macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CnzConv.
CnzConv	
ƀ	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).
,OutConsoleId =*OutConsoleId Addr*	*OutConsoleId Addr:* RX-type address or register (2) - (12).
,OutConsoleName =*OutConsoleName Addr*	*OutConsoleName Addr:* RX-type address or register (2) - (12).
,ConsoleStatus =*ConsoleStatus Addr*	*ConsoleStatus Addr:* RX-type address or register (2) - (12).
,ConsoleType =*ConsoleType Addr*	*ConsoleType Addr:* RX-type address or register (2) - (12).
,ConsoleSubType =*ConsoleSubType Addr*	*ConsoleSubType Addr:* RX-type address or register (2) - (12).
,Sysname =*Sysname Addr*	*Sysname Addr:* RX-type address or register (2) - (12).
,SMCS_LU =*SMCS_LU Addr*	*SMCS_LU Addr:* RX-type address or register (2) - (12).
,SubsysOwnerName =*SubsysOwnerName Addr*	*SubsysOwnerName Addr:* RX-type address or register (2) - (12).
,SubsysASID =*SubsysASID Addr*	*SubSysASID Addr:* RX-type address or register (2) - (12).
,RtnCode =*RtnCode Addr*	*RtnCode Addr:* RX-type address or register (2)-(12).
,RsnCode =*RsnCode Addr*	*RsnCode Addr:* RX-type address or register (2)-(12).

Parameters

The parameters are explained as follows:

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. Console subtypes are only returned for the console types of MCS, EMCS, and Special. HMCS is a valid subtype for the console type of MCS. SYSCON is a valid subtype for the console type of EMCS. Internal, Instream, Unknown, and JES3 are valid subtypes for the console type of Special. 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.

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:

Return Code	Reason Code	Meaning and Action
0	N/A	**Name**: CnzConvRc0_Ok **Meaning**: The input console name or ID was found and the applicable requested data was returned. **Action**: None.
4	xxxxxxxx	**Name**: CnzConvRc4_ConditionallyOK **Meaning**: The request completed successfully with an exception. **Action**: Examine the reason code to determine how to proceed.
4	xxxx0401	**Name**: CnzConvRsn401_IdNotFound **Meaning**: The console ID in InConsoleId is not associated with any console. **Action**: 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.
4	xxxx0402	**Name**: CnzConvRsn402_NameNotFound **Meaning**: The console name in InConsoleName is not associated with any console. **Action**: 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.

Return Code	Reason Code	Meaning and Action
4	xxxx0403	**Name**: CnzConvRsn403_NameIsReserved **Meaning**: The input console name is a reserved console name. **Action**: 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.
8	xxxxxxxx	**Name**: CnzConvRc8_SpecificationError **Meaning**: An error was detected in the CnzConv parameter list. None of the requested data has been returned. **Action**: Correct the CnzConv parameter list. Examine the reason code to determine how to proceed.
8	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	**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	**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	**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	**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	xxxxxxxx	**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.

Return Code	Reason Code	Meaning and Action
C	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	**Name**: CnzConvRsnC02_IncorrectEnv **Meaning**: The CnzConv service was invoked in an incorrect environment. **Action**: Invoke the CnzConv service in the correct environment.
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).

```
NZCONVEX CSECT
NZCONVEX AMODE 31
NZCONVEX RMODE ANY
*********************************************************************
*                                                                   *
*         REGISTER ASSIGNMENTS                                      *
*                                                                   *
*********************************************************************
REG0     EQU   0                       REGISTER 0
REG1     EQU   1                       REGISTER 1
REG2     EQU   2                       REGISTER 2
REG3     EQU   3                       REGISTER 3
REG4     EQU   4                       REGISTER 4
REG5     EQU   5                       REGISTER 5
REG6     EQU   6                       REGISTER 6
REG7     EQU   7                       REGISTER 7
REG8     EQU   8                       REGISTER 8
REG9     EQU   9                       REGISTER 9
REG10    EQU   10                      REGISTER 10
REG11    EQU   11                      REGISTER 11 - DYNAMIC DATA AREA
```

```
REG12    EQU   12                      REGISTER 12
REG13    EQU   13                      REGISTER 13
REG14    EQU   14                      REGISTER 14
REG15    EQU   15                      REGISTER 15
         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
               CONSOLETYPE=OUTCONSOLETYPE,                             X
               CONSOLESUBTYPE=OUTCONSOLESUBTYPE,                       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 *
*********************************************************************
*     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'MCSY13E0  '  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
RSV000001          DS     CL1
OUTSYSNAME         DS     CL8
OUTSMCS_LU         DS     CL8
OUTSUBSYSOWNERNAME DS     CL8
OUTSUBSYSASID      DS     H
CNZCONVRETURNCODE  DS     F
CNZCONVREASONCODE  DS     F
  CNZCONV          MF=(L,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:

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

Parameters

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

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CnzConv.
CnzConv	
ƀ	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).
[,OutConsoleId = *OutConsoleId Addr*]	*OutConsoleId Addr:* RX-type address or register (2) - (12).
[,OutConsoleName = *OutConsoleName Addr*]	*OutConsoleName Addr:* RX-type address or register (2) - (12).
[,ConsoleStatus = *ConsoleStatus Addr*]	*ConsoleStatus Addr:* RX-type address or register (2) - (12).
[,ConsoleType = *ConsoleType Addr*]	*ConsoleType Addr:* RX-type address or register (2) - (12).
[,ConsoleSubType = *ConsoleSubType Addr*]	*ConsoleSubType Addr:* RX-type address or register (2) - (12).
[,Sysname = *Sysname Addr*]	*Sysname Addr:* RX-type address or register (2) - (12).
[,SMCS_LU = *SMCS_LU Addr*]	*SMCS_LU Addr:* RX-type address or register (2) - (12).
[,SubsysOwnerName = *SubsysOwnerName Addr*]	*SubsysOwnerName Addr:* RX-type address or register (2) - (12).

Syntax	Description
[,SubsysASID = *SubsysASID Addr*]	*SubSysASID Addr:* RX-type address or register (2) - (12).
[,RtnCode=*RtnCode Addr*]	*RtnCode Addr:* RX-type address or register (2)-(12).
[,RsnCode=*RsnCode Addr*]	*RsnCode Addr:* RX-type address or register (2)-(12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Parameters

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

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

Chapter 29. CNZTRKR — Tracking interface macro

Description

The CNZTRKR macro can be used to issue the tracking facility. IBM recommends using macro GTZTRACK instead. This service allows programmers to record events of interest. For more information about the tracking facility, *z/OS MVS Diagnosis: Tools and Service Aids*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	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 you issue 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:

Field	Description
TRPL_Acro	The TRPL acronym
TRPL_Version	The current version level of the parameter list. The CNZTRPL mapping macro contains the current version level in TRPL_K_Curr_Version.
TRPL_Track_Info	Text that describes the occurrence of this instance. This text can be 1 - 28 characters in length. Any EBCDIC value is allowed. Although you might use displayable characters because in-displayable characters can 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.
TRPL_Track_Data	Four bytes of data associated with this track instance. Zero is a valid value. The DISPLAY OPDATA operator command displays this value as a hexadecimal number.

Field	Description
TRPL_Violators_Addr	While optional, this field contains the address of where the tracked event occurred. Possible to the address in which the service issues CNZTRKR returns. If set to zero, the tracking facility attempts to determine the address of the event, but might 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.

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 information into any register unless they are using it in register notation for a particular parameter, or by using it as a base register.

Output register information

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

Register
Contents

0
Reason code.

1
Used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
Return code.

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

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

Syntax

The CNZTRKR macro is written as follows:

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

Syntax	Description
ƀ	One or more blanks must precede CNZTRKR.
CNZTRKR	
ƀ	One or more blanks must follow CNZTRKR.
register	*register*: General-purpose register (2) - (12).
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 requests to ABEND the program that issues the tracking service, the task is ABENDed with an ABEND code of X'E77' and a user-defined reason code. See the SETGTZ DEBUG operator command for further details.

Return and reason codes

When the CNZTRKR macro returns control to your program, register 15 contains the return code and register 0 contains a reason code, if the return code is not zero. Mapping macro CNZTRPL provides names for the return and reason codes. Refer to Chapter 102, “GTZTRACK macro — GTZ Track,” on page 679 for codes that are not listed in CNZTRPL.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 is reached. **Action**: See the description of message GTZ0004E.
04	08	**Meaning**: The request was to record an instance but the tracking facility is not active. **Action**: See the description of reason code xxxx0401 of macro service GTZTRACK.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0C	*xx*	**Meaning**: There was an error with the TRPL parameter list. **Action**: None.
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**: An error attempts to access the TRPL parameter list. The TRPL address which you provided might not be valid or pointed to storage that the CNZTRKR service might 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 might 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 might not be established. **Action**: If this instance is important to be recorded, you can reissue the request. Serialization is 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.
10	10	**Meaning**: GTZTRACK rejected the tracking request. **Action**: Notify your system programmer.

Chapter 30. 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 28, "CnzConv -- Convert console name and ID ," on page 147.

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 the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) 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:

Environmental factor	Requirement
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

  CONVVRSN
  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.

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:

Register
Contents

0
Reason code

1
Address of the CONV parameter list

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CONVCON macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CONVCON.
CONVCON	
ƀ	One or more blanks must follow CONVCON.
register	*register:* General purpose register (2) - (12).
list name	*list name:* RX-type address.
,RTNCODE=ReturnCode	*ReturnCode:* RX-type address or register (2)-(12).
,RSNCODE=ReasonCode	*ReasonCode:* RX-type address or register (2)-(12).

Parameters

The parameters are explained as follows:

register
list name
Contains the address (*register*) or the name (*list name*) of the CONV parameter list.

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:

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: The input console is active and the area ID (if specified) is syntactically valid. **Action**: None.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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.
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.

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

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   CONVVRSN,CONVRID         PUT VERSION INTO PARAMETER LIST
         OI    CONVFLGS,CONVPFLD        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
```

Chapter 31. 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* and *z/Architecture Principles of Operation* 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. The leap second offset is not considered when determining the output.

Environment

The requirements for the caller are:

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

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

Syntax	Description
,TIMETYPE=DEC	**Default**: TIMETYPE=DEC.
,TIMETYPE=BIN	
,TIMETYPE=MIC	
,DATETYPE=YYDDD	
,DATETYPE=YYYYDDD	**Default**: DATETYPE=YYYYDDD.
,DATETYPE=DDMMYYYY	
,DATETYPE=MMDDYYYY	
,DATETYPE=YYYYMMDD	
,OFFSET=*offset value*	*offset value*: RX-type address or register (2) - (12).
	Default: OFFSET=X'0000000F'.

Parameters

The parameters are explained as follows:

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 June 4, 2185, which corresponds to the end of the second epoch (the corresponding ETOD value would have a value less than x'02' in the first byte).

,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 HHMMSSthmiju0000, where

HH
is hours, based on a 24-hour clock

MM
is minutes

SS
is seconds

t
is tenths of a second

h
is hundredths of a second

m
is milliseconds

i
is ten-thousandths of a second

j
is hundred-thousandths of a second

u
is microseconds.

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:

Parameter	Format of input date
YYDDD	0CYYDDDF
YYYYDDD	0YYYYDDD
DDMMYYYY	DDMMYYYY
MMDDYYYY	MMDDYYYY
YYYYMMDD	YYYYMMDD

Where:

0C
is the century - 00 represents 19YY, 01 represents 20YY

F
is a sign to enable the date to be unpacked

YY
is the last two digits of the year

YYYY
is the year

DDD
is the day of the year (Julian date)

DD
is the day of the month

MM
is the month of the year

,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 5. Return Codes for the CONVTOD Macro

Return Code	Meaning and Action
00 (00)	**Meaning**: Successful completion. **Action**: None.
0C (12)	**Meaning**: Unsuccessful completion. CONVTOD encountered an unexpected error. **Action**: Record the return code and supply it to the appropriate IBM support personnel.
10 (16)	**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.
14 (20)	**Meaning**: Unsuccessful completion. The time, date, or offset parameter value was not valid. **Action**: Verify that the input parameters have been initialized correctly. Avoid specifying a date or time that occurs after the second epoch (the corresponding ETOD value would have a value greater than x'01' in the first byte).

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:

```
         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'0000500D'               -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:

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

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CONVTOD.
CONVTOD	
ƀ	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	
,DATETYPE=YYDDD	
,DATETYPE=YYYYDDD	**Default**: DATETYPE=YYYYDDD.
,DATETYPE=DDMMYYYY	
,DATETYPE=MMDDYYYY	
,DATETYPE=YYYYMMDD	

Syntax	Description
,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:

,MF=(E,*list addr*)
Specifies the execute form of the CONVTOD macro. *list addr* specifies the area that the system uses to store the parameters.

Chapter 32. 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:

Environmental factor	Requirement
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 can 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 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:

Register	Contents
13	The address of a 72-byte save area

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:

Register
Contents

0
Contains the cell pool id. The cell pool ID is never zero.

1
Used as a work register by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

When control returns to the caller from CPOOL GET, the GPRs contain:

Register
Contents

0
Used as work registers for the system.

1
For an UNCOND request or a successful COND request, contains the address of the obtained cell. For an unsuccessful COND request, contains a zero.

2-4
If REGS=SAVE is specified, the registers remain unchanged. Otherwise, used as work registers by the system.

5-13
If LINKAGE=SYSTEM, REGS=SAVE, or COND REGS=USE is specified, the registers remain unchanged. Otherwise, the registers are used as work registers by the system.

14-15
Used as work registers by the system.

When control returns to the caller from CPOOL FREE, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2-3
If REGS=SAVE is specified, the registers remain unchanged. Otherwise, used as work registers by the system.

4-13
Unchanged

14-15
Used as work registers by the system.

When control returns to the caller from CPOOL DELETE, the GPRs contain:

Register
Contents

0-1
Used as a work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

When control returns to the caller from CPOOL LIST, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as work registers by the system.

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

Register
Contents

0
For CPOOL GET,COND, unchanged. Otherwise, used as a work register by the system.

1
For CPOOL GET,COND, unchanged. Otherwise, used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
For CPOOL GET,COND or CPOOL FREE, unchanged. Otherwise, used as a work register by the system.

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede CPOOL.
CPOOL	
␣	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
,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.

Syntax	Description
,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 ,REGS=USE	**Default:** REGS=SAVE
,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.
- 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 might be more than what is expected and might 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=QWORD
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.

Note:

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.

,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 is 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=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 is 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.

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 registers 2 - 12 are to be saved for a GET or FREE request.

,REGS=SAVE
The registers are saved in a 72-byte user-supplied save area pointed to by register 13. The user-supplied save area must not be within the storage area to be freed.

,REGS=USE
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 might 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 - 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, for example, words 1-3, in the work area to return information to your program:

- Word 1 contains the return code. See “Return and reason codes” on page 186 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*.

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 - 7) of the work area used to return information to the calling program.

Table 6. Hexadecimal Return Codes for CPOOL LIST	
Return Code	**Meaning and Action**
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.
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 to contain 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 to specify 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
```

```
* 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CPOOL.
CPOOL	
ƀ	One or more blanks must follow CPOOL.
BUILD	
,PCELLCT=*primary cell count*	*cell count*: Symbol or decimal number.

Syntax	Description
	Note: PCELLCT must be specified on either the list or the execute form of the macro.
,SCELLCT=*secondary cell count*	**Default:** PCELLCT
,CSIZE=*cell size*	*cell size*: Symbol or decimal number.
	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	
,LOC=24	**Default:** LOC=RES
,LOC=31	
,LOC=(31,31)	
,LOC=(31,64)	
,LOC=(31,PAGEFRAMESIZE1MB)	
,LOC=RES	
,LOC=(RES,31)	
,LOC=(RES,64)	
,KEY=*key number*	*key number*: Decimal numbers 0-15.
	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 or A-type address.

Syntax	Description
,MF=L	

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CPOOL.
CPOOL	
ƀ	One or more blanks must follow CPOOL.
BUILD	
,PCELLCT=*primary cell count*	*cell count*: Symbol, decimal number, or register (0), (2) - (12).
	Note: PCELLCT must be specified on either the list or the execute form of the macro.
,SCELLCT=*secondary cell*	**Default:** PCELLCT
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).

Syntax	Description
	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	
,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).
,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 33. 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:

Environmental factor	Requirement
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:

  Register
  Contents

  13
  The address of an 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Performance implications

None.

Syntax

The CPUTIMER macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CPUTIMER.
CPUTIMER	
ƀ	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 could not be performed. The error routine executes in the addressing mode of the issuer of the CPUTIMER 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 7. Return and Reason Codes for the CPUTIMER Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The function was performed. **Action**: None.
04	**Meaning**: Program error. The function was not performed because the user-specified area was not on a doubleword boundary. **Action**: Ensure that the address of the area for the return of the CPU time is on a doubleword boundary.
08	**Meaning**: Program error. The function was not performed because the user supplied an invalid address. **Action**: Verify that the supplied return area address is valid.
10	**Meaning**: System error. The function was not performed because a machine check occurred. **Action**: Retry the request.
14	**Meaning**: System error. The function was not performed because a program check occurred. **Action**: Retry the request.

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 34. 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:

Environmental factor	Requirement
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:

Register
Contents

0
The run length encoding algorithm. Specify either 0 or 1.

13
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.

Output register information for SERVICE=QUERY

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

Register
Contents

0
A value of 1 to indicate that the run length encoding algorithm will be used.

1
The length of the work area required by the algorithm. This value might be zero if the service does not require a work area.

2-13
Unchanged.

14
Used as a work register by the system.

15
Return code.

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Input register information for SERVICE=COMPRESS

Before issuing the CSRCESRV macro with the SERVICE=COMPRESS parameter, the caller must ensure that the following GPRs contain the specified information:

Register
Contents

1
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.

2
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.

3
The length of the uncompressed input data block.

4
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.

5
The length of the output data block.

13
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.

Output register information for SERVICE=COMPRESS

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

Register
Contents

0-1
Unchanged

2
The 31-bit address of the byte following the last input byte processed

3
The number of bytes of uncompressed data not processed

4
The 31-bit address of the byte following the last output byte

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Input register information for SERVICE=EXPAND

Before issuing the CSRCESRV macro with the SERVICE=EXPAND parameter, the caller must ensure that the following GPRs contain the specified information:

Register
Contents

1
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.

2
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.

3
The length of the compressed input data block.

4
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.

5
The length of the output data block.

13
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.

Output register information for SERVICE=EXPAND

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

Register
Contents

0-1
Unchanged

2
The 31-bit address of the byte following the last input byte processed

3
The number of bytes of compressed data not processed

4
The 31-bit address of the byte following the last output byte

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CSRCESRV macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSRCESRV.
CSRCESRV	
ƀ	One or more blanks must follow CSRCESRV.
SERVICE=QUERY SERVICE=COMPRESS SERVICE=EXPAND	
,VECTOR=*(reg)*	*reg*: register (2) - (12).

Parameters

The parameters are explained as follows:

SERVICE=QUERY
SERVICE=COMPRESS
SERVICE=EXPAND
Specifies the requested service.

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 8. Return Codes for SERVICE=QUERY

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The requested algorithm is supported. This means that both compression and expansion are supported. **Action**: None.
04	**Meaning**: The requested algorithm is supported only for data expansion. **Action**: None.
0C	**Meaning**: Program error. The requested algorithm is not supported by this level of MVS. **Action**: Specify the appropriate input value and rerun the program.
10	**Meaning**: Program error. The algorithm number was negative. **Action**: Specify the appropriate input value and rerun the program.

Table 9. Return Codes for SERVICE=COMPRESS

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: All input data was compressed. **Action**: None.

Table 9. Return Codes for SERVICE=COMPRESS (continued)

Hexadecimal Return Code	Meaning and Action
04	**Meaning**: Program error. Not all input data was compressed 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 uncompressed data processed, or issue the macro with a larger output area.
0C	**Meaning**: Program error. Either the input or output length was negative. **Action**: Inspect the contents of the GPRs to determine which value is in error. Specify the appropriate input values and rerun the program.

Table 10. Return Codes for SERVICE=EXPAND

Hexadecimal Return Code	Meaning and Action
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 35. CSRCMPSC — Compress and expand data

Description

The CSRCMPSC macro performs the following functions:

- Compresses data
- Expands previously-compressed data
- Simulates expansion of data compressed using entropy encoding, for expanding data on a machine that does not have the hardware supporting entropy encoding (The IHAFACL bit FACL_EntropyEncoding will be off on such a machine).

Compression of data using entropy encoding is not supported by CSRCMPSC. Expansion of data using entropy encoding without simulation (using the CMPSC instruction) is not supported by CSRCMPSC.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, PSW key 8-15
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
Amode:	• AMODE 31 is supported for all options. • AMODE 64 is supported for expansion of data compressed using entropy encoding (Huffman compression).
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. All parameters may reside in storage above 2G when the caller is AMODE 64.

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.

See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the mapping macros.

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 standard save area in the primary address space, unless requesting expansion of an area compressed using entropy encoding, in which case this register need not be set.

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:

Register
Contents

13
0 which designates the primary address space unless requesting expansion of an area compressed using entropy encoding, in which case this register need not be set.

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:

Register
Contents

n
The ALET with which the system is to access the CSRYCMPS area. For primary mode callers, the ALET should be 0.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The standard form of CSRCMPSC is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSRCMPSC.
CSRCMPSC	
ƀ	One or more blanks must follow CSRCMPSC.
CBLOCK=*comp block*	*comp block*: RS-type address, or register (1) - (12).
,RETCODE=*rc*	*rc*: RS-type address, or register (2) - (12).
	Default: No return code processing.

Parameters

The parameters are explained as follows:

CBLOCK=*comp block*
A required input/output parameter specifying 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 or DSECT CMPSCEE in macro CSRYCMPS; see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for the CSRYCMPS macro.

Use CMPSCEE only when requesting expansion of an area compressed using entropy encoding, for which both bits CMPSCEE_EntropyEncoding and CMPSCEE_Expand must be on. The CBLOCK area provides the information needed by, and provided on return by, the CSRCMPSC service. It should begin on a fullword boundary when using DSECT CMPSC or on a doubleword boundary when using DSECT CMPSCEE.

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

RETCODE=*rc*
An optional output parameter specifying 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.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

Do not specify RETCODE with MF=M.

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.

Hexadecimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Meaning and Action
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.
34	**Meaning**: Program error. Entropy descriptors for Entropy Encoding are not valid. **Action**: Fix the dictionary.
38	**Meaning**: Program Error. The input block indicates compression and Entropy Encoding. That is not supported. **Action**: Avoid requesting compression for Entropy Encoding.
3C	**Meaning**: Program error. The dictionary, as indicated by the offset to the entropy descriptors, is not a valid size. **Action**: Fix the dictionary or the offset.

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
         CSRYCMPS ,
```

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,EDICTADDR
         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
 .
 .
         DS    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
         CSRYCMPS ,
```

Example 3

Expand a data area for entropy encoding. Note the following:

- The expansion dictionary must be aligned on a page boundary and must be followed immediately by the 16 2-byte entropy descriptors.
- The offset to the entropy descriptors divided by 128 must be in bits 52-60 of the expansion dictionary address.
- Assume that this code is AMODE 64 and has issued SYSSTATE AMODE64=YES.

```
         LA    2,MYCBLOCK               Get address of
parm

         USING
CMPSCEE,2

         XC    CMPSCEE(CMPSCEE_LEN),CMPSCEE Clear
block

         OI
CMPSCEE_FLAGS_BYTE1,CMPSCEE_EntropyEncoding

         OI    CMPSCEE_FLAGS_BYTE2,CMPSCEE_Expand Do
expansion

         LG
3,EDICTADDR

         STG   3,CMPSCEE_DICTADDR       Set dictionary
address

         LG
3,EXPADDR

         STG   3,CMPSCEE_TARGETADDR     Set expansion
area

         LG
3,EXPLEN

         STG   3,CMPSCEE_TARGETLEN      Set expansion
length

         LG
3,COMPADDR

         STG   3,CMPSCEE_SOURCEADDR     Set compression
area

         LG
3,COMPLEN

         STG   3,CMPSCEE_SOURCELEN      Set compression
length

         LA
3,WORKAREA

         STG   3,CMPSCEE_WORKAREAADDR   Set workarea
address
```

```
         CSRCMPSC
CBLOCK=CMPSCEE

         DROP
2

 .

 .

         DS    0D        Align parameter on word
boundary

MYCBLOCK DS    (CMPSCEE_LEN)CL1  CBLOCK
Parameter

EXPADDR  DS    AD        Output "To" (expansion)
area

EXPLEN   DS    D         Length of "To"
area

COMPADDR DS    AD        Input "From" (compression)
area

COMPLEN  DS    D         Length of "From"
area

EDICTADDR DS   AD        Address of expansion
dictionary

*                        with offset of entropy
descriptors

*                        divided by 128 in bits
52-60

         DS    0D        Doubleword align
workarea

WORKAREA DS    CL256     Work
area


CSRYCMPS ,
```

Example 4

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.

```
.
.
        LAE   2,MYCBLOCK              Set address *and* ALET
        CSRCMPSC CBLOCK=(2)           Issue operation
.
.
```

Chapter 36. 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:

Environmental factor	Requirement
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
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRC4ACT	,(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 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 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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 CSRC4ACT or CSRC4EXP).
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 37. 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:

Environmental factor	Requirement
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:

```
*  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 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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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 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 cell pool anchor.

,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 doubleword variable containing the cell size in this pool. You can use any positive binary or hexadecimal number as the cell size.

,return_code)
When CSRC4BLD completes, the fullword variable specified by *return_code* contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRC4BLD service returns control to your program, GPR 15 (and *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
00	00	**Meaning**: The operation was successful. **Action**: None.
18	24	**Meaning**: Program error. The anchor address is not valid. **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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
44	68	**Meaning**: Program error. The cell size is not valid: it cannot be negative or 0. **Action**: Specify a positive value for the cell size.

Chapter 38. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Syntax	Description
CALL CSRC4CON	,(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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 CSRC4ACT or CSRC4EXP).
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.
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 39. 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:

Environmental factor	Requirement
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
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRC4DAC	,(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 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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 being passed is for the right cell pool.

Chapter 40. 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:

Environmental factor	Requirement
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
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

Some callers depend on 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.

Syntax	Description
CALL CSRC4DIS	,(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 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 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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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**: Call CSRC4DAC to deactivate the extent before calling CSRC4DIS to disconnect the cell storage for the extent.
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 41. 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:

Environmental factor	Requirement
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
*
```

```
*
*  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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRC4EXP	,(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:

,(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_addr
Specifies the doubleword variable containing the address of the extent. The extent must begin on a quadword boundary.

,extent_size
Specifies the doubleword variable containing the size of the extent.

,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.

,area_size
Specifies the doubleword variable containing the length (binary or hexadecimal) of the storage area for the cells.

,extent_num
When CSRC4EXP completes, the fullword variable specifying *extent_num* contains the number of the extent to be connected. You will use this number on subsequent CALLs.

,return_code)
When CSRC4EXP completes, the fullword variable specifying *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 *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Chapter 42. 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:

Environmental factor	Requirement
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
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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:

Environmental factor	Requirement
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:
*
CSRC4_SAVEAREA_LEN     EQU     216
```

```
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

,(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 CSRC4FR1 is to free.

,return_code
When CSRC4FR1 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.

ABEND codes

None.

Return and reason codes

When the CSRC4FR1 service returns control to your program, GPR 15 (and *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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:

Environmental factor	Requirement
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
*
*
*  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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Chapter 45. 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:

Environmental factor	Requirement
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:
```

```
*
CSRC4_POOL_NAME_LEN     EQU     8
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

,(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
When CSRC4GET completes, the doubleword variable specified by *cell_addr* contains the address of the cell that CSRC4GET allocated.

,return_code)
When CSRC4GET completes, the fullword variable specified by *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 *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 46. 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:

Environmental factor	Requirement
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
*
*
*  Length of the user-supplied savearea:
*
CSRC4_SAVEAREA_LEN     EQU     216
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

,(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
When CSRC4GT1 completes, the doubleword variable specified by *cell_addr* contains the address of the cell that CSRC4GT1 allocated.

,return_code
When CSRC4GT1 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.

ABEND codes

None.

Return and reason codes

When the CSRC4GT1 service returns control to your program, GPR 15 (and *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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 additonal output parameter to return the address of the extent containing the obtained cell.)

Environment

The requirements for the caller are:

Environmental factor	Requirement
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
*
*
*  Length of the user-supplied savearea:
*
CSRC4_SAVEAREA_LEN     EQU     216
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRC4GT2	,(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
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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.
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 48. 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:

Environmental factor	Requirement
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:

```
*  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 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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRC4QCL	,(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 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:

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

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.
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 49. 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:

Environmental factor	Requirement
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:
*
CSRC4_POOL_NAME_LEN     EQU     8
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRC4QEX	,(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:

,(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 the service will query.

,status
When CSRC4QEX completes, the doubleword 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 CSRC4QEX completes, the doubleword variable specified for *extent_addr* contains the address of the extent.

,extent_len
When CSRC4QEX completes, the doubleword variable specified for *extent_len* contains the length of the extent, in bytes.

,area_addr
When CSRC4QEX completes, the doubleword variable specified for *area_addr* contains the address of cell storage.

,area_size
When CSRC4QEX completes, the doubleword variable specified for *area_size* contains the size of cell storage for the extent.

,total_cells
When CSRC4QEX completes, the doubleword variable specified for *total_cells* contains the total number of cells associated with the extents.

,avail_cells
When CSRC4QEX completes, the doubleword variable specified for *avail_cells* contains the total number of cells associated with the specified extent that are available for allocation.

,return_code)
When CSRC4QEX completes, the fullword variable specified for *return_code* contains the return code.

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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 50. 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:

Environmental factor	Requirement
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
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in 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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

,(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.

,user_name
When CSRC4QPL completes, the variable specified by *user_name* contains the name on the CSRC4BLD service that created the cell pool.

,cell_size
When CSRC4QPL completes, the doubleword variable specified by *cell_size* contains the size of each cell at the time the cell pool was created.

,total_cells
When CSRC4QPL completes, the doubleword variable specified by *total_cells* contains the total number of cells associated with the extents.

,avail_cells
When CSRC4QPL completes, the doubleword variable specified by *avail_cells* contains the total number of cells in active extents that are available for allocation.

,number_extents
When CSRC4QPL completes, the doubleword 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 CSRC4QPL completes, the fullword variable specified by *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 *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Chapter 51. 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:

Environmental factor	Requirement
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:

Register
Contents

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, CSRC4RFR ignores the value.

GPR 0
The doubleword address of the cell you want freed.

GPR 1
The doubleword anchor address.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in low 32 bits. High 32 bits are used as a work area by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRC4RFR	

Parameters

See “Input register information” on page 278.

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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 52. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

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, CSRC4RF1 ignores the value.

GPR 0
The doubleword address of the cell you want freed.

GPR 1
The doubleword anchor address.

GPR 13
The address of a 216-byte save area that your program provides. The system does not change the first 8 bytes of this area.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code in low 32 bits. High 32 bits are used as a work area by the system.

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRC4RF1	

Parameters

See “Input register information” on page 282.

ABEND codes

None.

Return and reason codes

When the CSRC4RF1 service returns control to your program, GPR 15 contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 53. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

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, CSRC4RGT ignores the value.

GPR 1
The doubleword anchor address.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the allocated cell.

2-13
Unchanged.

14
Used as a work register by the system.

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 ARs contain:

Register
Contents

0
Used as a work register by the system.

1-14
Unchanged.

15
Used as a work register by the system.

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRC4RGT	

Parameters

See “Input register information” on page 286.

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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

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, CSRC4RG1 ignores the value.

GPR 1
The doubleword anchor address.

GPR 13
The address of a 216-byte save area that your program provides. The system does not change the first 8 bytes of this area.

Output register information

When control returns to the caller, the 64-bit general purpose registers (GPRs) contain:

Register
Contents

0
Used as a work register by the system.

1
Address of the allocated cell.

2-13
Unchanged.

14
Used as a work register by the system.

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 ARs contain:

Register
Contents

0
Used as a work register by the system.

1-14
Unchanged.

15
Used as a work register by the system.

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRC4RG1	

Parameters

See “Input register information” on page 290.

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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 55. CSREVW — View an object and sequentially access it

Description

Call the CSREVW 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, CSREVW 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 CSREVW 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 CSREVW 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 CSREVW, 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 CSREVW 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:

Environmental factor	Requirement
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.

Input register information

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

Register
Contents

13
The address of a standard 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Some callers depend on 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.

Syntax	Description
CALL CSREVW	,(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:

,(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.

,reason_code)
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 89, “DIV — Data-in-virtual,” on page 489).

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00000000	00000000	**Meaning**: The operation was successful. **Action**: None.
00000004	00000165	**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 reason code *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 reason code *nnnn*.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 56. 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:

Environmental factor	Requirement
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:

Register
Contents

13
The address of a standard 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
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:

DDNAME
The object is an existing (OLD) VSAM linear data set allocated to the file whose DDNAME is specified by *object_name*.

DSNAME
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.

TEMPSPACE
The object is a temporary data object. Window services will delete the object when your program issues CSRIDAC END.

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.

,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.

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 89, “DIV — Data-in-virtual,” on page 489).

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00000000	00000000	**Meaning**: The operation was successful. **Action**: None.
00000004	xxxxnnnn	**Meaning**: Program error or environmental error. The operation was successful; however, data-in-virtual issued a warning. 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**.
00000008	00000118	**Meaning**: Environmental error. The system could not obtain enough storage to create a hiperspace for the temporary object or the scroll area. **Action**: 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.
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 *rrrr* is the return code from dynamic allocation. The value *nnnn* 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.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 57. CSRL16J — Transfer control with all registers intact

Description

Call the CSRL16J service to transfer control with all registers intact 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:

Environmental factor	Requirement
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 L16J or L16J1 as defined in macro CSRYL16J. For more information, see Defining the entry characteristics of the target routine in *z/OS MVS Programming: Callable Services for High-Level Languages*.
- 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:

```
*********************************************************************/
*                      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:

Register
Contents

13
Address of a standard 18 word save area

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When CSRL16J returns control to the caller because of an error, the access registers (ARs) contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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 11. Return Codes for the CSRL16J Service

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Successful completion. The calling program does not receive this return code because it indicates that the target routine received control. **Action**: None.
04	**Meaning**: The value specified in the L16JVERSION field of the L16J data area was not zero. The L16JVERSION field must contain a value of zero. **Action**: 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.
08	**Meaning**: The calling program was not in 31-bit addressing mode, which is required. **Action**: Make sure the calling program is in 31-bit addressing mode.
0C	**Meaning**: 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. **Action**: 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.
10	**Meaning**: The value specified in field L16JLENGTH in CSRYL16J was less than the actual length of the data area. **Action**: Make sure that the value in the L16JLENGTH field reflects the actual length of the data area.
18	**Meaning**: The PSW provided in field L16JPSW of CSRYL16J specified an ASC mode that is not valid. **Action**: In the L16JPSW field, specify either primary or AR ASC mode.

Chapter 58. 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:

Environmental factor	Requirement
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 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 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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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.

,extent_num
Identifies 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 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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 59. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRPBLD	,(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 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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
00	00	**Meaning**: The operation was successful. **Action**: None.
18	24	**Meaning**: Program error. The anchor address is not valid. **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	**Meaning**: Program error. The cell size is not valid: it cannot be negative or 0. **Action**: Specify a positive value for the cell size.

Chapter 60. 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:

Environmental factor	Requirement
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.

Input register information

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

Output register information

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Syntax	Description
CALL CSRPCON	,(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.

,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. CSRPCON 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 CSRPCON completes, the variable specified by *return_code* contains the return code.

ABEND codes

None.

Return and reason codes

When the CSRPCON service returns control to your program, GPR 15 (and *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 CSRPACT or CSRPEXP).
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.
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 61. 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:

Environmental factor	Requirement
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:
*
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

After calling CSRPDAC, you can still free (or return) cells, but you cannot get (or allocate) any others for this extent.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 62. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

4
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

Some callers depend on 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.

Syntax	Description
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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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**: Call CSRPDAC to deactivate the extent before calling CSRPDIS to disconnect the cell storage for the extent.
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 CSRPDIS 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 63. 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:

Environmental factor	Requirement
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 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 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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

,(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.

,extent_addr
Specifies the variable containing the address of the extent. The extent must begin on a doubleword boundary.

,extent_size
Specifies the variable containing the size of the extent.

,area_addr
Specifies the variable containing starting address of the cell storage area. The starting address of this area must be consistent with any boundary requirements that you might have.

,area_size
Specifies the variable containing the length (binary or hexadecimal) of the storage area for the cells.

,extent_num
When CSRPEXP completes, the variable specifying *extent_num* contains the number of the extent to be connected. You will use this number on subsequent CALLs.

,return_code)
When CSRPEXP completes, the variable specifying *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 *return_code*) contains one of the following return codes:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Chapter 64. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

,(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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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:

Environmental factor	Requirement
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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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:

,(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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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 additonal input parameter for the address of the extent containing the cell to be freed.)

Environment

The requirements for the caller are:

Environmental factor	Requirement
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.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRPFR2	,(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 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
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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Chapter 67. 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:

Environmental factor	Requirement
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
*
*
```

Restrictions

None.

Input register information

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

Output register information

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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:

Environmental factor	Requirement
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
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRPGT1	,(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:

,(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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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. 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 CSRPGET service provides the same function but slightly slower performance. CSRPGT2 is preferred over CSRPGT1 when using multiple extents, as CSRPGT2 has an additonal output parameter to return the address of the extent containing the obtained cell.)

Environment

The requirements for the caller are:

Environmental factor	Requirement
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
```

```
*
*
```

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRPGT2	,(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 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 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:

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

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.
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 70. 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:

Environmental factor	Requirement
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.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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.

,anchor_addr
Specifies the variable containing the address of the 64-byte anchor.

,cell_addr
Specifies the variable containing the address of the cell the service will query.

,cell_avail
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

,extent_num
When CSRPQCL completes, the variable specified for *extent_num* contains the number of the extent that contains the specified cell.

,return_code)
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:

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

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.
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 71. 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:

Environmental factor	Requirement
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.

Input register information

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

Output register information

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRPQEX	,(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:

,(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.

,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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 72. 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:

Environmental factor	Requirement
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:
*
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 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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL CSRPQPL	,(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:

,(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.

,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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Chapter 73. 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:

Environmental factor	Requirement
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:

Register
Contents

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, 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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRPRFR	

Parameters

See “Input register information” on page 374.

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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 74. 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:

Environmental factor	Requirement
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:

Register
Contents

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, CSRPRFR1 ignores the value.

GPR 0
The address of the cell you want freed.

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRPRFR1	

Parameters

See “Input register information” on page 378.

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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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.

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 75. 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:

Environmental factor	Requirement
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 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 CSRPRGT service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

Register
Contents

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:

Register
Contents

0
Used as a work register by the system

1
Address of the allocated cell

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0
Used as a work register by the system

1-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRPRGT	

Parameters

See “Input register information” on page 382.

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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 76. CSRPRGT1 — Allocate a cell from a cell pool (register interface)

Description

Call the CSRPRGT1 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. CSRPRGT1 allocates cells from the lowest- to highest-numbered active extents, and within each extent, from the lowest to highest cell address. (The CSRPRGT service provides the same function but slightly slower performance.)

Environment

The requirements for the caller are:

Environmental factor	Requirement
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 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 CSRPRGT1 service, the caller must ensure that the following access registers (ARs) and general purpose registers (GPRs) contain the specified information:

Register
Contents

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:

Register
Contents

0
Used as a work register by the system

1
Address of the allocated cell

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0
Used as a work register by the system

1-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

Write the call as shown on the syntax diagram.

Syntax	Description
CALL CSRPRGT1	

Parameters

See “Input register information” on page 386.

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:

Hexadecimal Return Code	Decimal Return Code	Meaning and Action
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 77. 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:

Environmental factor	Requirement
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:

Register
Contents

13
The address of a standard 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Contains the reason code.

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

Write the CALL as shown on the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
CALL 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 89, “DIV — Data-in-virtual,” on page 489).

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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*.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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.

Chapter 78. 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:

Environmental factor	Requirement
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 CSRSCOT.

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:

Register
Contents

13
The address of a standard 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
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.

,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 *z/OS MVS 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 89, “DIV — Data-in-virtual,” on page 489).

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00000000	00000000	**Meaning**: The operation was successful. **Action**: None.
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'.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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*.
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 79. 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:

Environmental factor	Requirement
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:

Register
Contents

13
The address of a standard 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
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 89, “DIV — Data-in-virtual,” on page 489).

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00000000	00000000	**Meaning**: The operation was successful. **Action**: None.
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'.
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*.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0000002C	00000004	**Meaning**: Program error or system 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 80. 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:

Environmental factor	Requirement
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.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

Syntax	Description
CALL CSRSI	(Request ,Infoarealen ,Infoarea ,Returncode)

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)
    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

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.

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 the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).
- If you are coding in C, use include file CSRSIC.

The following table describes the return codes, shown in decimal.

Return Code (decimal)	Equate Symbol Meaning and Action
00	**Equate Symbol**: CSRSI_SUCCESS **Meaning**: The CSRSI service completed successfully. All information requested was returned. **Action**: Check the si00validityflags field to determine the validity of each returned area.
04	**Equate Symbol**: CSRSI_STSINOTAVAILABLE **Meaning**: 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. **Action**: None required.
08	**Equate Symbol**: CSRSI_SERVICENOTAVAILABLE **Meaning**: Environmental error: The CSRSI service is not available on this system. **Action**: Avoid calling the CSRSI service unless running on a system on which it is available.
12	**Equate Symbol**: CSRSI_BADREQUEST **Meaning**: 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. **Action**: Correct the parameter.
16	**Equate Symbol**: CSRSI_BADINFOAREALEN **Meaning**: User error: The Infoarealen parameter did not match the length of the area required to return the requested information. **Action**: Correct the parameter.

Return Code (decimal)	Equate Symbol Meaning and Action
20	**Equate Symbol**: CSRSI_BADLOCK **Meaning**: User error: The service was called while holding a system lock other than CPU. LOCAL/CML, or CMS. **Action**: Avoid calling in this environment.

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:

```
#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;


#ifdef __cplusplus
   ??>
#endif

#ifndef __cplusplus
#define csrsi_byaddr(Request, Flen, Fptr, Rcptr)                    \
??&>                                                                \
 struct CSRSI_PSA* CSRSI_pagezero = 0;                              \
   CSRSI_pagezero->CSRSI_cvt->CSRSI_cvtcsrt->CSRSI_addr             \
           (Request,Flen,Fptr,Rcptr);                               \
??>;
#endif
  ??>;
struct CSRSI_CSRT ??&>
   unsigned char CSRSI_csrt_filler1  ??(48??);
   CSRSI_calltype* CSRSI_addr;
```

```
  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
                    CVTOSLVL 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_cvtcsrsi : 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  si11v1cpcmanufacturer??(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  si11v1cpctype??(4??); /* The 4-character (0-9) EBCDIC
                                     type identifier of the V1 CPC.
                                                                  */
  unsigned char  _filler2??(12??);  /* Reserved                   */

  unsigned char  si11v1cpcmodel??(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  si11v1cpcsequencecode??(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  si11v1cpcplantofmanufacture??(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.
                                                                  */
```

```
        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
```

```
/*********************************************************************/
/* si22v2 represents the output for a V2 CPC when information        */
/* is requested about the set of CPUs                                */
/*********************************************************************/
               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
                                    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.               */
```

```
  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
                                                                  */
??> si22v2lcpuc;                  /* Characteristics             */
unsigned int   si22v2totallcpucount          : 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   si22v2configuredlcpucount     : 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   si22v2standbylcpucount        : 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.
                                                                 */
```

```
     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
                                    si22v2lcpudedicated.)        */
```

```
      unsigned int   si22v2sharedlcpucount           : 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   si22v3dbtotallcpucount            : 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   si22v3dbconfiguredlcpucount       : 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.
                                                                  */
```

```
      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.                    */
```

```
      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                     */
??> si22v3db;
/*********************************************************************/
/* 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  si22v3dbe??(8??);    /* Array of entries. Only the number
                                    indicated by si22v3dbcount
                                    are valid                       */
  unsigned char  _filler5??(3552??); /* Reserved                    */
??> si22v3;

#define si22v3dbcount      si22v3dbcountfield._si22v3dbcount


/*********************************************************************/
/* SI00 represents the "starter" information. This structure is      */
/* part of the information returned on every CSRSI request.          */
/*********************************************************************/
```

```
typedef struct ??&>
  char           si00cpcvariety;    /* SI00CPCVariety_V1CPC_MACHINE,
                                       SI00CPCVariety_V2CPC_LPAR, or
                                       SI00CPCVariety_V3CPC_VM       */
    struct ??&>
              int   _si00validsi11v1  : 1; /* si11v1 was requested and
                                    the information returned is valid
                                                                     */
              int   _si00validsi22v1  : 1; /* si22v2 was requested and
                                    the information returned is valid
                                                                     */
              int   _si00validsi22v2  : 1; /* si22v2 was requested and
                                    the information returned is valid
                                                                     */
              int   _si00validsi22v3  : 1; /* si22v3 was requested and
                                    the information returned is valid
                                                                     */
              int   _filler1          : 4; /* Reserved             */
    ??> si00validityflags;
  unsigned char  _filler2??(2??);   /* Reserved                     */
  unsigned char  si00pccacpid??(12??); /* PCCACPID value for this CPU
                                                                     */
  unsigned char  si00pccacpua??(2??); /* PCCACPUA value for this CPU
                                                                     */
  unsigned char  si00pccacafm??(2??); /* PCCACAFM value for this CPU
                                                                     */
  unsigned char  _filler3??(4??);   /* Reserved                     */
  unsigned char  si00lastupdatetimestamp??(8??); /* Time of last STSI
                                       update, via STCK             */
  unsigned char  _filler4??(32??);  /* Reserved                     */
 ??> si00;

#define si00validsi11v1         si00validityflags._si00validsi11v1
```

```
#define si00validsi22v1         si00validityflags._si00validsi22v1
#define si00validsi22v2         si00validityflags._si00validsi22v2
#define si00validsi22v3         si00validityflags._si00validsi22v3

/*********************************************************************/
/* siv1 represents the information returned when V1CPC_MACHINE       */
/* data is requested                                                 */
/*********************************************************************/

typedef struct ??&>
  si00 siv1si00;                                  /* Area mapped by
                                  struct si00                       */
  si11v1 siv1si11v1;                                /* Area
                                  mapped by struct si11v1           */
  si22v1 siv1si22v1;                                /* Area
                                  mapped by struct si22v1           */
??> siv1;
```

```
/*********************************************************************/
/* siv1v2 represents the information returned when V1CPC_MACHINE     */
/* data and V2CPC_LPAR data is requested                             */
/*********************************************************************/

typedef struct ??&>
  si00 siv1v2si00;                                /* Area mapped by
                                  by struct si00                    */
  si11v1 siv1v2si11v1;                              /* Area
                                  mapped by struct si11v1           */
  si22v1 siv1v2si22v1;                              /* Area
                                  mapped by struct si22v2           */
  si22v2 siv1v2si22v2;                              /* Area
                                  mapped by struct si22v2           */
??> siv1v2;

/*********************************************************************/
/* siv1v2v3 represents the information returned when V1CPC_MACHINE   */
/* data, V2CPC_LPAR data and V3CPC_VM data is requested              */
/*********************************************************************/

 typedef struct ??&>
  si00 siv1v2v3si00;                              /* Area
                                  mapped by struct si00             */
  si11v1 siv1v2v3si11v1;                            /* Area
                                  mapped by struct si11v1           */
  si22v1 siv1v2v3si22v1;                            /* Area
                                  mapped by struct si22v1           */
  si22v2 siv1v2v3si22v2;                            /* Area
                                  mapped by struct si22v2           */
  si22v3 siv1v2v3si22v3;                            /* Area
                                  mapped by struct si22v3           */
 ??> siv1v2v3;

/*********************************************************************/
/* siv1v3 represents the information returned when V1CPC_MACHINE     */
/* data and V3CPC_VM data is requested                               */
/*********************************************************************/

typedef struct ??&>
  si00 siv1v3si00;                                /* Area mapped
                                  by struct si00                    */
  si11v1 siv1v3si11v1;                              /* Area
                                  mapped by struct si11v1           */
  si22v1 siv1v3si22v1;                              /* Area
                                  mapped by struct si22v1           */
  si22v3 siv1v3si22v3;                              /* Area
                                  mapped by struct si22v3           */
??> siv1v3;
```

```
/*********************************************************************/
/* siv2 represents the information returned when V2CPC_LPAR          */
/* data is requested                                                 */
/*********************************************************************/

typedef struct ??&>
  si00 siv2si00;                  /* Area mapped by
```

```
                                      struct si00                  */
  si22v2 siv2si22v2;                /* Area
                                      mapped by struct si22v2      */
??> siv2;

/*********************************************************************/
/* siv2v3 represents the information returned when V2CPC_LPAR        */
/* and V3CPC_VM data is requested                                    */
/*********************************************************************/

typedef struct ??&>
  si00 siv2v3si00;                  /* Area mapped
                                      by struct si00               */
  si22v2 siv2v3si22v2;              /* Area
                                      mapped by struct si22v2      */
  si22v3 siv2v3si22v3;              /* Area
                                      mapped by struct si22v3      */
??> siv2v3;

/*********************************************************************/
/* siv3 represents the information returned when V3CPC_VM            */
/* data is requested                                                 */
/*********************************************************************/

typedef struct ??&>
  si00 siv3si00;                    /* Area mapped by
                                      struct si00                  */
  si22v3 siv3si22v3;                /* Area
                                      mapped by struct si22v3      */
??> siv3;
```

```
/*********************************************************************/
/*       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
```

Chapter 81. 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:

Environmental factor	Requirement
Minimum authorization:	Problem state and PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
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:

Register
Contents

13
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.

Before issuing the CSRUNIC macro in AR-ASC mode, the caller must ensure that the following access registers (ARs) contain the specified information:

Register
Contents

13
ALET of the 72-byte save area pointed to by GPR 13.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CSRUNIC macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSRUNIC.
CSRUNIC	
ƀ	One or more blanks must follow CSRUNIC.
FUNCTION=MVCLU	

Syntax	Description
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.

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
A required parameter that designates the function to be performed.

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.

FUNCTION=CUUTF
indicates to process a CUUTF operation.

FUNCTION=CUTFU
indicates to process a CUTFU operation.

,PBLOCK=*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 12. Return Codes for the CSRUNIC Macro

Return Code	Equate Symbol	Meaning and Action
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.

Table 12. Return Codes for the CSRUNIC Macro (continued)		
Return Code	**Equate Symbol**	**Meaning and Action**
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).
1C	UNIC_CLCLU_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_TP_RC_Valid	**Meaning**: the operand is a valid packed number. **Action**: None required.
4	UNIC_TP_RC_SignNotValid	**Meaning**: The sign of the operand was not valid. All the digits were valid. **Action**: None required.
8	UNIC_TP_RC_DigitNotValid	**Meaning**: One or more digits of the operand were not valid. The sign was valid. **Action**: None required.
0C	UNIC_TP_RC_SignDigitNotValid	**Meaning**: The sign and one or more digits of the operand were not valid. **Action**: None required.
1C	UNIC_TP_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_PKA_RC_OK	**Meaning**: The pack operation completed successfully. **Action**: None required.

Table 12. Return Codes for the CSRUNIC Macro (continued)

Return Code	Equate Symbol	Meaning and Action
14	UNIC_PKA_RC_SourceLengthNotValid	**Meaning**: The length of the source 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_PKA_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_PKU_RC_OK	**Meaning**: The pack operation completed successfully. **Action**: None required.
14	UNIC_PKU_RC_SourceLengthNotValid	**Meaning**: The length of the source operand exceeded 64 bytes (that is, the LengthMinusOne exceeded 63). **Action**: Avoid calling CSRUNIC REQUEST=PKU for an operand longer than 64 bytes.
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.

Table 12. Return Codes for the CSRUNIC Macro (continued)

Return Code	Equate Symbol	Meaning and Action
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.
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).

Table 12. Return Codes for the CSRUNIC Macro (continued)

Return Code	Equate Symbol	Meaning and Action
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.
0	UNIC_TROT_RC_TestCharNotFound	**Meaning**: The translation completed. The test character was not found. **Action**: None required.
4	UNIC_TROT_RC_TestCharFound	**Meaning**: The test character was found. The operation ended at that point. **Action**: None required.
1C	UNIC_TROT_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_TROO_RC_TestCharNotFound	**Meaning**: The translation completed. The test character was not found. **Action**: None required.
4	UNIC_TROO_RC_TestCharFound	**Meaning**: The test character was found. The operation ended at that point. **Action**: None required.
1C	UNIC_TROO_RC_WorkareaNotAligned	**Meaning**: The workarea provided was not on a doubleword boundary. **Action**: Make sure that the workarea is on a doubleword boundary.

Table 12. Return Codes for the CSRUNIC Macro (continued)		
Return Code	**Equate Symbol**	**Meaning and Action**
0	UNIC_TRE_RC_TestCharNotFound	**Meaning**: The translation completed. The test character was not found. **Action**: None required.
4	UNIC_TRE_RC_TestCharFound	**Meaning**: The test character was found. The operation ended at that point. **Action**: None required.
1C	UNIC_TRE_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_CUUTF_RC_SourceExhausted	**Meaning**: All unicode characters in the source were converted to their UTF-8 equivalents. **Action**: None required.
4	UNIC_CUUTF_RC_TargetExhausted	**Meaning**: The target operand did not have enough room to hold the UTF-8 equivalents of all of the source unicode characters. **Action**: Provide a larger target area.
1C	UNIC_CUUTF_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_CUTFU_RC_SourceExhausted	**Meaning**: All UTF-8 characters in the source were converted to their unicode equivalents. **Action**: None required.
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 82. 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 CSREVW, 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 CSREVW 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:

Environmental factor	Requirement
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:

Register
Contents

13
The address of a standard 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

Write the CALL as shown in the syntax diagram. You must code all parameters on the CALL statement in the order shown.

Syntax	Description
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.

,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 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.

,disposition
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.

,return_code
When CSRVIEW completes, *return_code* contains the return code. Define *return_code* as integer data of length 4.

,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 89, “DIV — Data-in-virtual,” on page 489).

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 83. 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:

Environmental factor	Requirement
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 CSVAPFAA mapping macro (see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary). For all other requests, you can optionally include the CSVAPFAA 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:

Register
Contents

13
For a QUERY request, the address of a standard 72-byte save area

Output register information

When control returns to the caller, the GPRs contain:

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:

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

Syntax	Description
ƀ	One or more blanks must precede CSVAPF.
CSVAPF	
ƀ	One or more blanks must follow CSVAPF.
	Valid parameters (Required parameters are underlined):
REQUEST=QUERY	DSNAME, VOLTYPE, VOLUME, RETCODE, RSNCODE
REQUEST=QUERYFORMAT	FORMAT
REQUEST=LIST	ANSAREA, ANSLEN, RETCODE, RSNCODE
,DSNAME=*libname*	*libname*: RS-type address or address in register (2) - (12).
,VOLTYPE=SMS	**Default**: VOLTYPE=SMS
,VOLTYPE=ANY,	VOLUME is required with VOLTYPE=ANY.
VOLUME=*volume*	*volume*: RS-type or address in register (2) - (12).
,FORMAT=*format*	*format*: RS-type address or address in register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: RS-type address or address in register (2) - (12).
,ANSLEN=*anslen*	*anslen*: RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

REQUEST=QUERY
REQUEST=QUERYFORMAT
REQUEST=LIST
Specifies the type of service to be performed on the list of APF-authorized program libraries. Specify one of the following:

QUERY
Determine if a particular library is in the APF list.

QUERYFORMAT
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.

LIST
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.

Note: The list will include those libraries that are defined or defaulted to be APF-authorized. The definition could be via IEAAPFxx 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.

,DSNAME=*libname*
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.

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 binary 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.

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* ".

,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.

,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.

,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.

,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 13. Return and Reason Codes for the CSVAPF Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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. **Action**: None.

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

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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” on page 431 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.
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.

Table 13. Return and Reason Codes for the CSVAPF Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
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.
10	xxxx1001	**Meaning**: System error. An internal error occurred. **Action**: Contact the system programmer. Provide the return code, the reason code, and the explanation of the error.

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    CVTOSLV1-CVTMAP(15),CVTDYAPF  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,APAA@                       Save answer area address
LAB1     DS    0H
         L     4,APAA@                       Get answer area address
         CSVAPF REQUEST=LIST,ANSAREA=(4),ANSLEN=APAALEN,              *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
         CLC   LRETCODE,=AL4(CSVAPFRC_OK)    Success?
         BE    LAB3                          Yes, process data
         CLC   LRETCODE,=AL4(CSVAPFRC_WARN) Warning?
         BNE   LAB2                          No, Process other return codes
         NC    LRSNCODE,=AL4(CSVAPFRSNCODEMASK)     Clear high order bits
         CLC   LRSNCODE,=AL4(CSVAPFRSNNOTALLDATARETURNED)      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,APAA@                       Save answer area address
         B     LAB1                          Re-do LIST request
LAB2     DS    0H                            Process other return codes
  .
  .
OLDLIST  DS    0H
* Current code to process static format APF list
  .
  .
         B     LAB9
LAB3     DS    0H
```

```
* New code to scan return information from CSVAPF
  .
  .
         L     4,APAA@
         L     3,APAALEN
         FREEMAIN RU,LV=(3),A=(4)  Release APAA
LAB9     DS    0H              End of processing
  .
  .
APAA@    DS    A               Address of APF answer area
APAALEN  DS    F               Length of APF answer area
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
         CSVAPFAA ,            Include CSVAPFAA mapping
```

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:

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

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

Parameters

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

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

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:

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

Syntax	Description
	Valid parameters (Required parameters are underlined):
REQUEST=QUERY	DSNAME, VOLTYPE, VOLUME, RETCODE, RSNCODE
REQUEST=LIST	ANSAREA, ANSLEN, RETCODE, RSNCODE
,DSNAME=*dsname*	*dsname*: RS-type address or register (2) - (12).
,VOLTYPE=SMS	**Default**: VOLTYPE=SMS
,VOLTYPE=ANY,	VOLUME is required with VOLTYPE=ANY.
VOLUME=*volume*	*volume*: RS-type or register (2) - (12).
,FORMAT=*format*	*format*: RS-type address, or register (2) - (12).
,ANSAREA=*ansarea*	*ansarea*: A-type address, or register (2) - (12).
,ANSLEN=*anslen*	*anslen*: A-type address, or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RS-type address, or register (2) - (12).
,MF=(E,*list addr,*COMPLETE)	**Default**: COMPLETE

Parameters

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

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

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

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

Chapter 84. 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 that is created by 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.

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

End of References

Typically, a path name returned from CSVINFO is prefixed by a slash (/); however, if that path name was returned in response to a load (BPX1LOD), exec (BPX1EXC) or spawn (BPX1SPN) call where the UNIX program was found in the current working directory, the path name will not be prefixed with a "/". To determine the full path name of the UNIX 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 path name as returned from BPX1CGW to prefix the path name returned by CSVINFO to determine the full path name of the UNIX program.

For information about the CSVMODI data area, see *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

Environment

Requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key See additional information under "Programming requirements" on page 442.
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. Other callers may not hold any locks.
Control parameters:	Must be in the primary address space

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:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

If 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:

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

Syntax	Description
ƀ	One or more blanks must follow CSVINFO.
FUNC=LPA	
FUNC=JPA,TCBADDR=*tcbaddr*	
FUNC=TASKLOAD,TCBADDR=*tcbaddr*	
FUNC=TASKALL,TCBADDR=*tcbaddr*	*tcbaddr*: RS-type address or address in register (2) - (12).
FUNC=RB,RBADDR=*rbaddr*	*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 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.

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 14. Return Codes for the CSVINFO Macro

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: Successful completion. **Action**: None.
4	**Meaning**: Successful completion. **Action**: None. There was no information for CSVINFO to return.
8	**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.
C	**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.
10	**Meaning**: Program error. A parameter specified for CSVINFO was inaccessible or not valid. **Action**: Correct the parameters and rerun the program.
14	**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.
18	**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.
1C	**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.
20	**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.
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.

Table 14. Return Codes for the CSVINFO Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
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:

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

Syntax	Description
ƀ	One or more blanks must follow CSVINFO.
MF=(L,*list addr*)	*list addr*: Symbol.
MF=(L,*list addr*, *attr*)	*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*)
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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVINFO.
CSVINFO	
ƀ	One or more blanks must follow CSVINFO.
FUNC=LPA	
FUNC=JPA,TCBADDR=*tcbaddr*	
FUNC=TASKLOAD,TCBADDR=*tcbaddr*	
FUNC=TASKALL,TCBADDR=*tcbaddr*	*tcbaddr*: RS-type address or address in register (2) - (12).

Syntax	Description
FUNC=RB,RBADDR=*rbaddr*	*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.
,USERDATA=NULL	**Default**: NULL
,COM=*com*	*com*: Comment text enclosed in single quotation marks.
,COM=NULL	**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*)	*list addr*: RS-type address.
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

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.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVINFO.
CSVINFO	
ƀ	One or more blanks must follow CSVINFO.
FUNC=LPA	
FUNC=JPA,TCBADDR=*tcbaddr*	
FUNC=TASKLOAD,TCBADDR=*tcbaddr*	
FUNC=TASKALL,TCBADDR=*tcbaddr*	*tcbaddr*: RS-type address or address in register (2) - (12).
FUNC=RB,RBADDR=*rbaddr*	*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.
,USERDATA=NULL	**Default**: NULL

Syntax	Description
,COM=*com*	*com*: Comment text enclosed in single quotation marks.
,COM=NULL	**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=(M,*list addr*)	*list addr*: RS-type address or register (2) - (12).
,MF=(M,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained under the standard form of the 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 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.

Chapter 85. 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, and 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:

Environmental factor	Requirement
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 can 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 can 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:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
Return code

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

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14-15
Used as a work register by the system.

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

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:

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

Syntax	Description
INEPNAME=*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 ,SEARCHMINOR=YES	**Default**: NO
,DIRLOAD=NO	**Default**: NO
,DIRLOAD=YES	
,OUTLENGTH=*length*	*length*: RS-type address or register (2) - (12).
,OUTLENGTH64=*length64*	*length64*: RS-type address or register (2) - (12).
,OUTEPNM=*entryname*	*entryname*: 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).

Syntax	Description
,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=*outpathname*	*outpathname*: RS-type address or register (2) - (12).
,OUTDSKEY=*outdskey*	*outdskey*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0 - 7
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

INEPNAME=*entryname*
INEPTKN=*ineptkn*
INADDR=*ptr name*
INADDR64=*ptr name64*

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.

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 INADDR, INADDR64, or INEPNAME, specifying the 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.

INADDR=*ptr name* specifies an address that CSVQUERY attempts to match to a loaded module. The address can be anywhere within the module.

INADDR64=*ptr name64* specifies an 8-byte address that CSVQUERY attempts to match to a loaded module. The address can be anywhere within the module.

You must specify one of the following mutually exclusive parameters: INEPNAME, INEPTKN, INADDR, or INADDR64.

,SEARCH=JPALPA
,SEARCH=JPA
,SEARCH=LPA

Specifies the type of search CSVQUERY is to perform.

JPALPA (the default) causes CSVQUERY to search the caller's job pack area. If the search fails, CSVQUERY searches the link pack area.

JPA causes CSVQUERY to search only the caller's job pack area.

LPA causes CSVQUERY to search only the link pack area.

,SEARCHMINOR=NO
,SEARCHMINOR=YES

Specifies whether to search for minor entry points. SEARCHMINOR is an optional parameter.

SEARCHMINOR=NO specifies that CSVQUERY is not to search for minor entry points. NO is the default.

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 INADDR parameter. Because the search is for the closest, CSVQUERY must check all entries.

,DIRLOAD=NO
,DIRLOAD=YES

Specifies whether to try using directed load information to resolve cases where the issuing module could not be determined, and would otherwise be returned as not found. DIRLOAD is an optional parameter. (A directed load module is a module that is directly loaded to a specified storage address).

DIRLOAD=NO specifies that CSVQUERY not try resolving not-found cases using directed load information. NO is the default.

DIRLOAD=YES specifies that CSVQUERY try resolving not-found cases using directed load information. A match on directed load information can return only the following outputs:

- OUTEPNM and OUTMJNM, which has the same value.
- OUTLOADPT
- OUTLOADPT64
- OUTPATHNAME

- OUTVALID
- OUTXATTR1

Note that a request with DIRLOAD=YES requires more processing time than one specifying or defaulting to DIRLOAD=NO.

When you specify DIRLOAD=YES, the CSVQUERY request can only resolve not-found cases when the following are true:

- The TRACKDIRLOAD parameter enabling system-wide tracking of directed load modules is specified in the PROGxx member of SYS1.PARMLIB.
- The module was added using z/OS system services, rather than being added without using intended programming interfaces.
- The directed load table still contains the information needed to determine the issuing module. this might not be the case if the table has wrapped in such a way that the old information needed has been replaced.
- The directed load occurred after above-2G storage use was enabled.
- For an address in common storage, the load of the module occurred from the primary address space where the CSVQUERY was issued.

When CSVQUESRY resolves a not-found case using the directed load information, it sets the X'00000000_00000001' bit in the XATTR1 output area.

When CSVQUERY cannot resolve a not-found case, if remains as not found.

,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 can 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 can 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*.

When the entry point address of the module is at least 2G, *entry addr* is set to X'7FFFFBAD'.

,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 OUTEPA64 with INADDR64, CSVQUERY returns the module's major entry point address in *entry addr64*.

,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.

When the load point address of the module is at least 2G, *outloadpt* is set to X'7FFFFBAD'.

,OUTLOADPT64=*outloadpt64*
Specifies an optional doubleword 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.

,OUTSP=*subpool*
Specifies an optional 1-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=*attr1*
Specifies an optional 1-byte variable where CSVQUERY returns the attributes of the module.

The bit settings have the following meanings:

Bit	Meaning When Set
0	End-of-memory deletion.
1	Loaded-to-global.
2	Reentrant.
3	Serially reusable.
4	Not loadable only.
5	Overlay format.
6	Alias
7	Not part of the programming interface.

,OUTATTR2=*attr2*
Specifies an optional 1-byte variable where CSVQUERY returns the attributes of the module.

The bit settings have the following meanings:

Bit	Meaning When Set
0	Authorized library.
1	Authorized program.
2	AMODE ANY.
3	AMODE 31.
5	Dynamic LPA module.

Bit	Meaning When Set
6	Page protected (only valid for dynamic LPA modules).
7	AMODE 64.

,OUTATTR3=*attr3*
Specifies an optional 1-byte variable where CSVQUERY returns the attributes of the module.

The bit settings have the following meanings:

Bit	Meaning When Set
0	Resident above 16 megabytes.
1	Job pack area resident.
2	PLPA resident.
3	MLPA resident.
4	FLPA resident.
5	CSA resident.
6-7	Not part of the programming interface.

,OUTXATTR1=*xattr*
Specifies an optional 8-byte variable where CSVQUERY returns extended attributes of the module.

The bit settings have the following meanings:

BYTE	BIT	Meaning When Set
0		Not part of the programming interface.
1		Not part of the programming interface.
2	1...	A RACF basic program.
2	.1..	A RACF main program.
3		Not part of the programming interface.
4		Not part of the programming interface.
5		Not part of the programming interface.
6		Not part of the programming interface.
7	1	The INADDR or INADDR64 address was resolved from an entry in the directed load 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.

Bit	Valid Field When Set
0	OUTLENGTH
1	OUTEPA
2	OUTEPNM
3	OUTMJNM
4	OUTSP
5	OUTATTR1

Bit	Valid Field When Set
6	OUTATTR2
7	OUTATTR3
8	OUTLOADPT
9	OUTPDATA
10	OUTPID
11	OUTEPTKN
12	OUTXTLST
13	OUTDIAG
14	OUTRTID
15	Not part of the programming interface.
16	OUTEPA64
17	OUTLOADPT64
18	OUTLENGTH64
19	OUTXTLST64
20	OUTXATTR1
21	OUTPATHNAME
22	OUTDSKEY
23-31	Not part of the programming interface.

,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:

Value
Meaning

'UNK '
Unknown provider

'LPA'
LPA

'PGMF'
Program fetch

'LLAF'
LLA

'AOSL'
AOS loader

'JPA'
JPA

,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 extent of the module that it has located. The first 4 bytes in the area should be initialized to 16, which is the total number of entries that can be returned.

On output, the second 4 bytes of the area contain 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 the extent address does not fit in 31 bits, a value of X'7FFFFBAD' is returned for the extent address, and a value of 1 is returned for the extent length. You can use the OUTXTLST64 parameter to get the full information.

If you specify SEARCHMINOR=YES and CSVQUERY finds a minor entry point, CSVQUERY returns the lengths of the module that contains the major entry point that is 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 extent of the module that it has located. The first 4 bytes in the area should be initialized to 16, which is the total number of entries that can be returned.

On output, the second 4 bytes of the area contain 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=*outpathname*
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
- INEPNAME
- OUTATTR1
- OUTATTR2
- OUTATTR3
- OUTEPA
- OUTEPNM
- OUTLENGTH
- OUTMJNM
- OUTSP
- OUTVALID
- SEARCH
- SEARCHMINOR

1

Version **1** introduces the following parameter:

- OUTLOADPT

2

Version **2** introduces the following parameters:

- INEPTKN
- OUTEPTKN
- OUTPDATA
- OUTPID

3

Version **3** introduces the following parameters:

- OUTDIAG
- OUTRTID
- OUTXTLST

5

Version **5** introduces the following parameters:

- INADDR64

- OUTEPA64
- OUTLENGTH64
- OUTLOADPT64
- OUTXATTR1
- OUTXTLST64

6
Version **6** introduces the following parameter: OUTPATHNAME

7
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:

Decimal Return Code	Meaning
00	CSVQUERY retrieves all the requested information.
04	CSVQUERY locates the specified module, but at least one requested output field is not valid.
08	CSVQUERY cannot locate the specified module.
12	CSVQUERY cannot obtain the lock(s) needed to process the request.
16	CSVQUERY encounters an unexpected error.
20	The requested function is not available on the system on which CSVQUERY is issued.

CSVQUERY - List form

Use the list form of the CSVQUERY macro to construct a nonexecutable parameter list.

Syntax

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

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

Syntax	Description
ƀ	One or more blanks must precede CSVQUERY.
CSVQUERY	
ƀ	One or more blanks must follow CSVQUERY.
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0 - 6
MF=(L,*list addr)*	*list addr*: Symbol.
MF=(L,*list addr,attr)*	*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:

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

Syntax	Description
b	One or more blanks must follow CSVQUERY.
	Valid parameters
INEPNAME=*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 ,SEARCH=JPA ,SEARCH=LPA	**Default**: JPALPA
,SEARCHMINOR=NO	**Default**: NO
,SEARCHMINOR=YES	
,OUTLENGTH=*length*	*length*: RS-type address or register (2) - (12).
,OUTLENGTH64=*length64*	*length64*: RS-type address or register (2) - (12).
,OUTEPNM=*entryname*	*entryname*: 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).

Syntax	Description
,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=*outpathname*	*outpathname*: RS-type address or register (2) - (12).
,OUTDSKEY=*outdskey*	*outdskey*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0 - 6
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RS-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE

Syntax	Description
,MF=(E,*list addr*,NOCHECK)	

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVQUERY.
CSVQUERY	
ƀ	One or more blanks must follow CSVQUERY.
	Valid parameters
INEPNAME=*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 ,SEARCH=JPA ,SEARCH=LPA	**Default**: JPALPA

Syntax	Description
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0 - 6
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,MF=(M,*list addr*)	*list addr*: RS-type address or register (2) - (12).
,MF=(M,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained under the standard form of the 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 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 86. 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:

Environmental factor	Requirement
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
RMODE:	Includes 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.
- DELETE is sensitive to SYSSTATE RMODE64=YES. When DELETE is issued with SYSSTATE RMODE64=YES, it may be invoked from a module at or above 2G.

Restrictions

None.

Input register information

None.

Output register information

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

Register
Contents

0
Address of the name or list entry that was supplied through the EP or DE keyword.

1-14
Unchanged.

15
Return code.

Syntax

The DELETE macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DELETE.
DELETE	
ƀ	One or more blanks must follow DELETE.
EP=*entry name*	*entry name*: Symbol.
EPLOC=*entry name addr*	*entry name addr*: RX-type address, or register (0) or (2) - (12).
DE=*list entry addr*	*list entry addr*: RX-type address, or register (0) or (2) - (12).
,RELATED=*value*	*value*: any valid macro keyword specification.

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 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:

Hexadecimal Return Code	Meaning
00	Successful completion of requested function
04	Requested module was not in storage, or an attempt was made to delete a system module by a caller not authorized to do so

Example

Remove a module (PGMTOVLY) from virtual storage.

```
DELETE EP=PGMTOVLY
```

Chapter 87. 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:

Environmental factor	Requirement
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

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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

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

Syntax	Description
(	
qname addr	*qname addr*: A-type address, or register (2) - (12).
,	
,rname addr	*rname addr*: A-type address, or register (2) - (12).
,	*rname length*: symbol, decimal digit, or register (2) - (12).
,rname length	**Note**: *rname length* must be coded if a register is specified for *rname addr*.
,	**Default**: STEP
,STEP	
,SYSTEM	
,SYSTEMS	
)	
,RET=NONE	**Default**: RET=NONE
,RET=HAVE	
,UCB=*ucb addr*	*ucb addr*: A-type address, or register (2) - (12).
	Note: Specify UCB only with SYSTEMS.
,LOC=BELOW	**DEFAULT**: LOC=BELOW
,LOC=ANY	
,RNL=YES	**Default**: RNL=YES
,RNL=NO	
,RELATED=*value*	*value*: any valid macro keyword specification.
,LINKAGE=SVC	**DEFAULT**: LINKAGE=SVC
,LINKAGE=SYSTEM	

Parameters

The parameters are explained as follows.

(
Specifies the beginning of the resource description.

qname addr
Specifies the address of an 8-character name. The name can contain any valid hexadecimal 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 topic of this chapter) to prevent interference from unauthorized programs.

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

,
,rname addr
Specifies the address of the name used together with *qname* 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 rname must be the same name specified for the resource in an ENQ or RESERVE macro.

,
,rname length
Specifies the length of the *rname*. 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 *rname*. 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 *rname* must be contained in the first byte at the *rname addr*.

,
,STEP
,SYSTEM
,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, *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=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* 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 1.

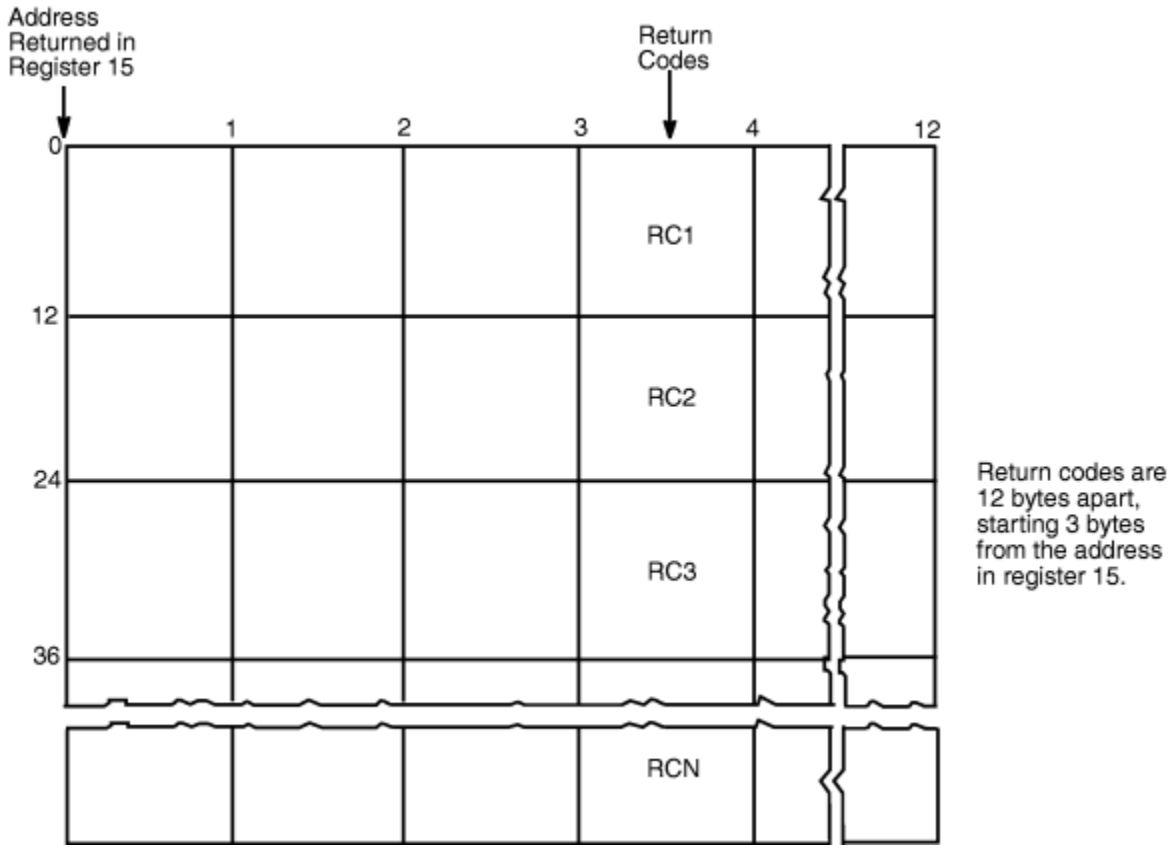


Figure 4. Return Code Area Used by DEQ

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 1.

Table 15. Return Codes for the DEQ Macro with the RET=HAVE Parameter

Hexadecimal Return Code	Meaning and Action
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 94, “ENQ — Request control of a serially reusable resource,” on page 573), if it has been assigned to the current task.

```
DEQ (MAJOR1,MINOR1,,STEP),RET=HAVE
```

Example 2

Unconditionally release control of the resources in Example 2 of ENQ. The length of the *rname* for the first resource is 3 characters and the length of the *rname* 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:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DEQ.
DEQ	
ƀ	One or more blanks must follow DEQ.
(	
qname addr	*qname addr*: A-type address.
,	*rname addr*: A-type address.
,rname addr	
,	*rname length*: symbol or decimal digit.
,rname length	
,	**Default**: STEP
,STEP	
,SYSTEM	
,SYSTEMS	
)	

Syntax	Description
,RET=NONE	**Default**: RET=NONE
,RET=HAVE	
,UCB=*ucb addr*	*ucb addr*: A-type address.
,LOC=BELOW	**Default**: LOC=BELOW
,LOC=ANY	
,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 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:

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

Syntax	Description
(	**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).
,	*rname addr*: RX-type address, or register (2) - (12).
,*rname addr*	
,	*rname length*: symbol, decimal digit, or register (2) - (12).
,*rname length*	
,	
,STEP	
,SYSTEM	
,SYSTEMS	
)	**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	**Default**: LOC=BELOW
,LOC=ANY	
,RNL=YES	
,RNL=NO	
,RELATED=*value*	*value*: any valid macro keyword specification.
,LINKAGE=SVC	**DEFAULT**: LINKAGE=SVC
,LINKAGE=SYSTEM	

Syntax	Description
,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.

Chapter 88. 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:

Environmental factor	Requirement
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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The DETACH macro is written as follows:

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

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

ABEND codes

The caller of DETACH might receive one of the following ABEND codes:

ABEND Code	Associated Reason Code
X'13E'	None
X'23E'	0, 4, 8
X'33E'	None
X'43E'	None
X'53E'	None

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

Return and reason codes

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

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Successful completion. **Action**: None.
04	**Meaning**: Environmental error. An incomplete subtask was detached with STAE=YES specified; DETACH processing successfully completed. **Action**: None required. However, you might take some action based upon your application.

Example 1

Remove the subtask from the address space. The address of the TCB is in the fullword labeled SAVEWORD.

```
DETACH SAVEWORD
```

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 89. 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:

Service
Function

IDENTIFY
Identifies you as a user of a data-in-virtual object.

ACCESS
Provides access to the data-in-virtual object.

MAP
Makes the data-in-virtual object addressable through your virtual window.

RESET
Releases changes made in your window since the last SAVE operation.

SAVE
Saves changed data that is in your window.

SAVELIST
Returns the addresses of the first and last changed pages in each range of changed pages within the window.

UNMAP
Eliminates the correspondence between the data-in-virtual object and your virtual window.

UNACCESS
Eliminates your access to the data-in-virtual object.

UNIDENTIFY
Ends your use of the data-in-virtual object.

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:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task

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

Programming requirements

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 IARVSERV macro to share data in virtual storage, you must follow several requirements; see the chapter about sharing data through IARVSERV in *z/OS MVS Programming: Assembler Services Guide*. 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:

Register
Contents

13
The address of an 18-word save area

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR 15 contains a nonzero return code; otherwise, used as a work register by the system

1
Used as a work register by the system

2-14
Unchanged

15
Return code

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

Register
Contents

0 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DIV.
DIV	
ƀ	One or more blanks must follow DIV.
	Valid parameters:
	(Underlined parameters are those that you must specify.)
IDENTIFY	ID, TYPE, DDNAME *or* STOKEN

Syntax	Description
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).
,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=*	
,SPAN=*addr*	*addr*: RX-type address, or register (2) - (12).

Syntax	Description
,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 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 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.

Encrypted linear VSAM data sets are supported. For more information, see *z/OS DFSMS Using Data Sets*.

,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.

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.

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.

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 16. Return and reason codes for the DIV macro

Reason code	Return code	Abend code	Meaning and action
none	00	—	**Meaning**: Successful completion. **Action**: None.
0001	none	08B	**Meaning**: Unknown service was requested. **Action**: None.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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. **Action**: Retry the request one or more times until the other service currently executing for this ID completes.
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.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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.
0017	0C	—	**Meaning**: 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.) **Action**: Retry the request. If the problem persists, record the return and reason code and supply it to the appropriate IBM support personnel.
0018	none	08B	**Meaning**: The caller does not have UPDATE access to the object. **Action**: None.
0019	none	08B	**Meaning**: A page to be saved or reset was in the page fixed state. **Action**: None.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
001A	04	—	**Meaning**: Program error. The specified range does not encompass any mapped area of the object. **Action**: None required. However, you might want to check that the specified range for this operation was correct.
001B	none	08B	**Meaning**: The virtual storage area specified to be unmapped is not currently mapped. **Action**: None.
001C	08	—	**Meaning**: Environmental error. The object cannot be accessed at the current time. **Action**: Retry the request one or more times until the operation succeeds.
001D	none	08B	**Meaning**: The accessed object is not at the correct control interval size. **Action**: None.
001E	none	08B	**Meaning**: The length of the ddname exceeds the maximum size allowed. **Action**: None.
001F	none	08B	**Meaning**: The caller's storage protect key is not the same as when IDENTIFY was invoked. **Action**: None.
0020	none	08B	**Meaning**: An ACCESS was attempted by a task that does not own the specified ID. **Action**: None.
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.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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.
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.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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.
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.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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.
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.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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.
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.
004E	08	—	**Meaning**: DIV could not obtain storage for the encryption buffers for a DIV SAVE requeset. **Action**: The local system area of the address space is out of storage. The address space is over-committed and should be scaled back.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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.
0055	none	08B	**Meaning**: For a DIV ACCESS request, the DIV invoker is not authorized to the ICSF service facility class. **Action**: Contact your SAF administrator for access to the ICSF service.
0056	none	08B	**Meaning**: For a DIV ACCESS request, the DIV invoker is not authorized to the key label associated with the DIV object. **Action**: Contact your SAF administrator for access to the key label of the DIV object.
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.

Table 16. Return and reason codes for the DIV macro (continued)

Reason code	Return code	Abend code	Meaning and action
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.
0809	08	—	**Meaning**: For a DIV ACCESS request, the Integrated Cryptographic Service Facility (ICSF) was not available to provide a key to process an encrypted DIV data set. **Action**: Restart ICSF and retry the request.
080A	08	—	**Meaning**: An encrypted data set was used for the DIV object. As part of a DIV ACCESS request, DIV tried to convert the key label into a key but failed. The key associated with the key label was not found. **Action**: Contact your security administrator. The return and reason codes from ICSF are RC=8, RSN=10012 (decimal). There might be a problem with the cryptographic key data set.
080B	08	—	**Meaning**: For a DIV ACCESS request, an environmental or system error occurred either with ICSF or the cryptographic co-processor. **Action**: Contact your security administrator.
080C	08	—	**Meaning**: For a DIV ACCESS request, ICSF was unable to process the request because the required hardware (co-processor) was not available. **Action**: Upgrade your system's hardware so that the required cryptographic co-processor is available.

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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DIV.
DIV	
ƀ	One or more blanks must follow DIV.
	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, LISTADDR, MF
UNMAP	ID, AREA, RETAIN, STOKEN
UNACCESS	ID
UNIDENTIFY	ID
,ID=*addr*	*addr*: A-type address
,AREA=*addr*	*addr*: A-type address

Syntax	Description
,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.

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:

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

Syntax	Description
,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*: RX-type address, or register (2) - (12).
,OFFSET=*	
,RETAIN=YES	**Default**: RETAIN=NO.
,RETAIN=NO	
,SIZE=*addr*	*addr*: RX-type address, or register (2) - (12).
,SIZE=*	
,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	
,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:

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

Syntax	Description
,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	**Default**: LOCVIEW=NONE
,LOCVIEW=NONE	
,MODE=READ	**Default**: None
,MODE=UPDATE	
,OFFSET=*addr*	*addr*: RX-type address, or register (2) - (12).
,OFFSET=*	
,RETAIN=YES	**Default**: RETAIN=NO
,RETAIN=NO	
,SIZE=*addr*	*addr*: RX-type address, or register (2) - (12).
,SIZE=*	
,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	
,MF=(M,*addr*)	See explanation of parameters if omitted.

Syntax	Description

Parameters

,MF=(M,*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.

Chapter 90. 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:

Environmental factor	Requirement
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:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

The DOM macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	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*
MSGLIST=*list addr*
Specifies the message numbers of messages to be deleted.

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.

Note: MSGLIST identification numbers of zero (0), while counted against the maximum allowed of sixty (60), are ignored and not processed in any way (i.e. not presented to any exits or the SSI and not sent to other systems).

,TOKEN=*addr*
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.

,COUNT=*addr*
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
```

Chapter 91. 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 92, "DSPSERV — Create, delete, and control hiperspaces," on page 535.

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 perform the following functions:

- 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).
- Back data space virtual pages with 1 M page frames, if possible (PAGEFRAMESIZE=1M).

On the DSPSERV macro, data spaces are identified through STOKENs. A STOKEN is a unique identifier of address spaces, data spaces, and hiperspaces.

Environment

The requirements for the caller are:

Environmental factor	Requirement
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)

Environmental factor	Requirement
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 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 the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

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:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede DSPSERV.
DSPSERV	
␣	One or more blanks must follow DSPSERV.
	Valid parameters (Required parameters are underlined.)
CREATE	STOKEN, NAME, TYPE, GENNAME, OUTNAME, BACK, BLOCKS, TTOKEN, ORIGIN, NUMBLKS, PAGEFRAMESIZE
RELEASE	STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE
DELETE	STOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
LOAD	STOKEN, BLOCKS, START
OUT	STOKEN, BLOCKS, START
,PAGEFRAMESIZE=4K	**Default:** PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1M	
,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	

Syntax	Description
,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,*init*)	*init*: Number up to 524288.
,BLOCKS=0	0 specifies the installation default size.
,BLOCKS=(0,*init-addr*)	**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	
,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=*numrange_addr*	*numrange_addr*: RS-type address or register (2) - (12).
,NUMRANGE=1	**Default**: NUMRANGE=1
,VAR=NO	**Default**: VAR=NO
,VAR=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0

Syntax	Description
,HIDEZERO=NO	**Default:** HIDEZERO=NO
,HIDEZERO=YES	
,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 IARVSERV 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.

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.

,PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1M
Specifies the size of the page frames to back the data space virtual pages.

,PAGEFRAMESIZE=4K
Backs data space virtual pages with 4 K page frames at first reference. This is the default value.

,PAGEFRAMESIZE=1M
Backs data space virtual pages with pageable 1 M page frames at first reference. If pageable 1 M page frames are not available at first reference, 4 K frames will be used.

,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 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.

,GENNAME=NO
,GENNAME=COND
,GENNAME=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 '*nCCCC*XYZ' 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.

GENNAME=NO
The system does not generate a name. You *must* supply a name unique within the address space. GENNAME=NO is the default.

GENNAME=COND
The system generates a unique name only if you supply a name that is already being used. Otherwise, the system uses the name that you supply.

GENNAME=YES
The system takes the name that you supply on the NAME parameter and makes it unique.

Note: The maximum number of system generated names is 99,999. If all system-generated names are used, DSPSERV reuses generated names from previously deleted data spaces or hiperspaces. If all system-generated names are in use for active data spaces or hiperspaces, DSPSERV fails with return code **8** and reason code **0012**. Before we reach the maximum number of system-generated names, the counter will not be reset to zero until all data spaces and hiperspaces within the address space are deleted. The generated names counter are reset to zero when the job is recycled.

If you want the system to return the unique name it generates, use the OUTNAME parameter.

,OUTNAME=*outname-addr*
Specifies the address of the 8-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.

,START=*start-addr*
Specifies the address of a 4-byte variable containing the beginning address of a block of storage in a data space. The address must be on a 4-kilobyte boundary. START is required on RELEASE requests.

,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 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 (*max-addr* or *max*) and the initial size of the data space (*init-addr* or *init.*). A block is a unit of 4K bytes. You cannot extend the data space beyond its maximum size.

max-addr specifies the address of a field that contains the maximum size of the data space to be created. *max* is the number of blocks (up to 524,288) to be used for the data space.

init-addr specifies the address of the initial size of the data space. *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 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.

BLOCKS=*size-addr* in MVS/SP3.1.0 is incompatible with BLOCKS=*(size-addr)* in MVS/SP3.1.0e and later releases in the case where *size-addr* is a register. If you coded BLOCKS=*(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 BLOCKS=*((register))* before you recompile.

,BACK=31
,BACK=64
Specifies whether the data space pages can be backed by real storage above 2 gigabytes when defined IOON (fixed).

BACK=31 specifies that the data space pages will be backed by frames below 2 gigabytes when defined IOON.

BACK=64 specifies that the data space pages will be backed by frames above or below 2 gigabytes when defined IOON.

,TTOKEN=*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.

,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.

,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*.

,INLIST=NO
,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.

,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.

,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.

,VAR=NO
,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.
- 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.

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

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,HIDEZERO=NO
,HIDEZERO=YES
An optional keyword input that specifies whether the system hides page 0 of the data space so that references to that page do not succeed. Regardless, the data space starts at the returned origin, and the number of blocks requested, if available, are allocated. HIDEZERO=NO is the default.

- HIDEZERO=NO indicates not to hide page 0.
- HIDEZERO=YES indicates to hide page 0. The returned origin indicates the lowest address that may be used which will be x'1000'. When PageFrameSize=1M is in effect:
  - The first segment is backed by 4K pages.
  - If performance is critical, avoid using any address below X'100000'.

,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:

Return Code	Reason Code	Meaning and Action
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 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	xx0005xx	**Meaning**: Program error. Creation of the data space would violate installation criteria. See the IEFUSI installation exit in *z/OS MVS Installation Exits*. **Action**: Check with your system programmer for local restrictions on the creation and use of data spaces.
08	xx0009xx	**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.

Return Code	Reason Code	Meaning and Action
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. **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	xx0007xx	**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:

Return Code	Reason Code	Meaning
00	—	**Meaning**: DSPSERV EXTEND completed successfully. **Action**: None.
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
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,RANGELST
         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
RANGELST DS   CL256                    STORAGE FOR MAX NUMBER OF RANGES
DRLMAP   DS   0F                       THIS CREATES A DSECT
         IARDRL                        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:

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede DSPSERV.
DSPSERV	
␣	One or more blanks must follow DSPSERV.
	Valid parameters (Required parameters are underlined.)
CREATE	STOKEN, NAME, TYPE, GENNAME, OUTNAME, BACK,
	BLOCKS, TTOKEN, ORIGIN, NUMBLKS
RELEASE	STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE
DELETE	STOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
LOAD	STOKEN, BLOCKS, START
OUT	STOKEN, BLOCKS, START
,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).

Syntax	Description
,GENNAME=NO	**Default**: GENNAME=NO
,GENNAME=COND	
,GENNAME=YES	
,OUTNAME=*outname-addr*	*outname-addr*: RX-type address or register (2) - (12).
,PAGEFRAMESIZE=4K	**Default:** PAGEFRAMESIZE=4K
,PAGEFRAMESIZE=1M	
,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,*init*)	*init*: Number up to 524288.
,BLOCKS=0	0 specifies the installation default size.
,BLOCKS=(0,*init-addr*)	**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	
,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=*numrange-addr*	*numrange-addr*: RS-type address or register (2) - (12).
,NUMRANGE=1	**Default**: NUMRANGE=1

Syntax	Description
,VAR=NO	**Default**: VAR=NO
,VAR=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0
,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 92. 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 91, "DSPSERV — Create, delete, and control data spaces," on page 519.

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:

Environmental factor	Requirement
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)

Environmental factor	Requirement
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 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 the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

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:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DSPSERV.
DSPSERV	
ƀ	One or more blanks must follow DSPSERV.
	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*	*stoken-addr*: RX-type address or register (2) - (12).
,TYPE=HIPERSPACE	
,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).

Syntax	Description
,BLOCKS=*(max,init)*	*init-addr*: 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.
,BLOCKS=(0,*init-addr*)	**Default for CREATE**: BLOCKS=0
,BLOCKS=(*size-addr*)	*size-addr*: RX-type address or register (2) - (12).
,BLOCKS=(*size*)	*size*: Number up to 524288.
,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).
,NUMRANGE=1	**Default**: NUMRANGE=1
,VAR=NO	**Default**: VAR=NO
,VAR=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0
,MF=S	

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 - 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.

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=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 '*nCCCC*XYZ' 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.

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.

Note: The maximum number of system generated names is 99,999. If all system-generated names are used, DSPSERV reuses generated names from previously deleted data spaces or hiperspaces. If all system-generated names are in use for active data spaces or hiperspaces, DSPSERV fails with return code **8** and reason code **0012**. Before we reach the maximum number of system-generated names, the counter will not be reset to zero until all data spaces and hiperspaces within the address space are deleted. The generated names counter are reset to zero when the job is recycled.

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.

,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 *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.

,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 *numblks-addr*.

,INLIST=NO
,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.

,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.

,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.

,VAR=NO
,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:

- 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.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=*plistver*

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

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form; in this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,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:

Return Code	Reason Code	Meaning and Action
00	—	**Meaning**: DSPSERV CREATE completed successfully. **Action**: None.

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

Return Code	Reason Code	Meaning and Action
00	—	**Meaning**: DSPSERV EXTEND completed successfully. **Action**: None.

Return Code	Reason Code	Meaning and Action
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.

```
         DSPSERV CREATE,NAME=HSPCNAME,STOKEN=HSPCSTKN,             X
               TYPE=HIPERSPACE,BLOCKS=HSPBLCKS,ORIGIN=HSPCORG
*        .
HSPCNAME DC   CL8'TEMP    '            HIPERSPACE NAME
HSPCSTKN DS   CL8                      HIPERSPACE STOKEN
HSPCORG  DS   F                        HIPERSPACE ORIGIN RETURNED
HSPCSIZE EQU  10000000
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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DSPSERV.
DSPSERV	
ƀ	One or more blanks must follow DSPSERV.
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION

Syntax	Description
,PLISTVER=*plistver*	*plistver*: 0
,MF=(L,*list addr)*	*list addr*: Symbol.
,MF=(L,*list addr,attr)*	*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*)
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:

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

Syntax	Description
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,TYPE=HIPERSPACE	
,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,*init*)	*init*: Number up to 524288.
,BLOCKS=0	0 specifies the installation default size.
,BLOCKS=(0,*init-addr*)	**Default for CREATE**: BLOCKS=0
,BLOCKS=(*size-addr*)	*size-addr*: RX-type address or register (2) - (12).
,BLOCKS=(*size*)	*size*: Number up to 524288.
,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*: RX-type address or register (2) - (12). Required with INLIST=YES
,NUMRANGE=*numrange-addr*	*numrange-addr*: RX-type address or register (2) - (12).
,NUMRANGE=1	**Default**: NUMRANGE=1

Syntax	Description
,VAR=NO	**Default**: VAR=NO
,VAR=YES	
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0
,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:

,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.

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

Chapter 93. 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.

Note:

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:

Environmental factor	Requirement
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

Environmental factor	Requirement
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:

Register
Contents

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

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede EDTINFO.
EDTINFO	
␣	One or more blanks must follow EDTINFO.
CHKGRPS	**Note:** At least one of these functions is required: CHKGRPS,
	CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR,
CHKUNIT	RTNNAMD, RTNDEVN, MAXELIG, RTNUNAFF. If more than
	one of these is specified, a comma must be coded between
RTNUNIT	each of the keywords.
RTNUCBA	**Note:** See the tables following this diagram for information on
	parameter usage with these functions.
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).

Syntax	Description
,DEVTYPE=*devtype addr*	*devtype addr*: RS-type address or register (2) - (12).
,SUBPOOL=*subpool addr*	*subpool addr*: RS-type address or register (2) - (12).
,UCBALIST=*ucbalist addr*	*ucbalist addr*: RS-type address or register (2) - (12).
,UCBLIST=*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	**Default**: DYNAMIC=NO
,DYNAMIC=NO	
,LOC=BELOW	**Default**: LOC=BELOW
,LOC=ANY	
,RANGE=ALL	**Default**: RANGE=3DIGIT
,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).

Syntax	Description
,RETCODE=*retcode addr*	*retcode addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).

The following tables show how the parameters may be specified with the CHKGRPS, CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR, RTNNAMD, RTNDEVN, MAXELIG, and RTNUNAFF functions.

The IOCTOKEN, RETCODE, and RSNCODE parameters are optional with any of the functions.

Parameters	CHKGRPS	CHKUNIT	RTNUNIT	RTNUCBA	RTNGRID
DEVCOUNT	required	required	not valid	not valid	not valid
DEVLIST	required	required	not valid	not valid	not valid
DEVSTAT	optional	optional	not valid	not valid	not valid
UNITNAME	not valid	UNITNAME or DEVTYPE required	not valid	UNITNAME or DEVTYPE required	not valid
DEVTYPE	not valid	DEVTYPE or UNITNAME required	required	DEVTYPE or UNITNAME required	not valid
SUBPOOL	not valid	not valid	not valid	optional	not valid
UCBALIST	not valid	not valid	not valid	required	not valid
UCBLIST	not valid	not valid	not valid	not valid	required
GRIDLIST	not valid	not valid	not valid	not valid	required
ATTRAREA	not valid	not valid	not valid	not valid	not valid
DEVCLASS	not valid	not valid	not valid	not valid	not valid
NAMELIST	not valid	not valid	not valid	not valid	not valid
DYNAMIC	not valid	not valid	not valid	not valid	not valid
LOC	not valid	not valid	not valid	not valid	not valid
RANGE	not valid	not valid	not valid	not valid	not valid
DEVNLIST	not valid	not valid	not valid	not valid	not valid
RECMODE	not valid	not valid	not valid	not valid	not valid
DENSITY	not valid	not valid	not valid	not valid	not valid
OUTUNIT	not valid	not valid	required	not valid	not valid
OUTDEV	not valid	not valid	not valid	not valid	not valid

Parameters	RTNATTR	RTNNAMD	RTNDEVN	MAXELIG	RTNUNAFF
DEVCOUNT	not valid	not valid	not valid	not valid	not valid
DEVLIST	not valid	not valid	not valid	not valid	not valid

Parameters	RTNATTR	RTNNAMD	RTNDEVN	MAXELIG	RTNUNAFF
DEVSTAT	not valid	not valid	not valid	not valid	not valid
UNITNAME	UNITNAME or DEVTYPE required	not valid	UNITNAME or DEVTYPE required	UNITNAME or DEVTYPE required	not valid
DEVTYPE	DEVTYPE or UNITNAME required	not valid	DEVTYPE or UNITNAME required	DEVTYPE or UNITNAME required	not valid
SUBPOOL	not valid	optional	not valid	not valid	not valid
UCBALIST	not valid	not valid	not valid	not valid	not valid
UCBLIST	not valid	not valid	not valid	not valid	not valid
GRIDLIST	not valid	not valid	not valid	not valid	not valid
ATTRAREA	required	not valid	not valid	not valid	not valid
DEVCLASS	not valid	required	not valid	not valid	not valid
NAMELIST	not valid	required	not valid	not valid	not valid
DYNAMIC	not valid	not valid	optional	not valid	not valid
LOC	not valid	not valid	optional	not valid	not valid
RANGE	not valid	not valid	optional	not valid	not valid
DEVNLIST	not valid	not valid	required	not valid	not valid
RECMODE	not valid	not valid	not valid	required	not valid
DENSITY	not valid	not valid	not valid	required	not valid
OUTUNIT	not valid	not valid	not valid	OUTUNIT or OUTDEV required	OUTUNIT or OUTDEV required
OUTDEV	not valid	not valid	not valid	OUTDEV or OUTUNIT required	OUTDEV or OUTUNIT required

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.

Note:

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 the obtain the actual UCB address from a given device number. See, *z/OS MVS Programming: Authorized Assembler Services Reference SET-WTO*, and *z/OS MVS Programming: Authorized Assembler Services Guide* for the UCBLOOK macro.

RTNGRID
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.

RTNATTR
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.

RTNNAMD
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.

RTNDEVN
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.

MAXELIG
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.

RTNUNAFF
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.

,DEVCOUNT=*devcount addr*
Specifies the fullword input field that contains the number of entries in the input device number list and the optional output device status list.

,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.

,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:

Byte
Contents

0
Length of the attribute area (X'0A'). You must fill in this byte prior to issuing EDTINFO.

1-2
Flags describing the unit name:

Bit
Meaning

0
If bit is on, the unit name is an esoteric group name.

1
If bit is on, the unit name is VIO-eligible.

2
Not part of the programming interface.

3
If bit is on, the unit name contains TP class devices.

4-15
Not part of the programming interface.

3
Number of device classes in the unit name.

4-7
Number of generic device types in the unit name.

8-9
Not part of the programming interface.

,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
,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.

,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.

,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 *ioctoken addr*, the caller is notified through a return code.

If the input specified by *ioctoken addr* is set to binary zeros, EDTINFO sets IOCTOKEN to the current MVS I/O configuration token.

,RECMODE=*recmode addr*
Specifies the address of an 8-bit input that indicates the recording mode.

,DENSITY=*density addr*
Specifies the address of an 8-bit input that indicates the density.

,OUTUNIT=*outunit addr*
,OUTDEV=*outdev addr*
Specifies the address of an 8-character field where EDTINFO returns the unit name (OUTUNIT=*outunit addr*) or specifies the address of a 4-character field where EDTINFO returns the 4-byte device type (OUTDEV=*outdev addr*).

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

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

Return and reason codes

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

Return Code	Meaning
00	The requested function or functions were performed and no reason code information has been returned.
04	The requested function or functions were performed and information has been returned, as explained by the hexadecimal reason code that accompanies this return code. The reason code is in GPR 0 (and in *rsncode addr*, if you coded RSNCODE). **Reason Code** **Meaning** **01** The input device numbers do not belong to the same group. **02** One or more of the input device numbers does not belong to the input unit name or device type. **03** 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.

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

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 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,RSNCODE=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 for further information.

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

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

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

Parameters

The parameters are explained as follows:

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

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

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

EDTINFO - Execute form

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

Syntax

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

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

Syntax	Description
CHKGRPS	**Note:** At least one of these functions is required: CHKGRPS,
	CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR,
CHKUNIT	RTNNAMD, RTNDEVN, MAXELIG, RTNUNAFF. If more than
	one of these is specified, a comma must be coded between
RTNUNIT	each of the keywords.
RTNUCBA	**Note:** See the tables following this diagram for information on
	parameter usage with these functions.
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).
,UCBALIST=*ucbalist addr*	*ucbalist addr*: RS-type address or register (2) - (12).
,UCBLIST=*ucblist addr*	*ucblist addr*: RS-type address or register (2) - (12).

Syntax	Description
,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	**Default**: DYNAMIC=NO
,DYNAMIC=NO	
,LOC=BELOW	**Default**: BELOW
,LOC=ANY	
,RANGE=ALL	**Default**: RANGE=3DIGIT
,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).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(E,*list addr*,NOCHECK)	

Syntax	Description

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.

Parameters	CHKGRPS	CHKUNIT	RTNUNIT	RTNUCBA	RTNGRID
DEVCOUNT	required	required	not valid	not valid	not valid
DEVLIST	required	required	not valid	not valid	not valid
DEVSTAT	optional	optional	not valid	not valid	not valid
UNITNAME	not valid	UNITNAME or DEVTYPE required	not valid	UNITNAME or DEVTYPE required	not valid
DEVTYPE	not valid	DEVTYPE or UNITNAME required	required	DEVTYPE or UNITNAME required	not valid
SUBPOOL	not valid	not valid	not valid	optional	not valid
UCBALIST	not valid	not valid	not valid	required	not valid
UCBLIST	not valid	not valid	not valid	not valid	required
GRIDLIST	not valid	not valid	not valid	not valid	required
ATTRAREA	not valid	not valid	not valid	not valid	not valid
DEVCLASS	not valid	not valid	not valid	not valid	not valid
NAMELIST	not valid	not valid	not valid	not valid	not valid
DYNAMIC	not valid	not valid	not valid	not valid	not valid
LOC	not valid	not valid	not valid	not valid	not valid
RANGE	not valid	not valid	not valid	not valid	not valid
DEVNLIST	not valid	not valid	not valid	not valid	not valid
RECMODE	not valid	not valid	not valid	not valid	not valid
DENSITY	not valid	not valid	not valid	not valid	not valid
OUTUNITL	not valid	not valid	required	not valid	not valid
OUTDEV	not valid	not valid	not valid	not valid	not valid

Parameters	RTNATTR	RTNNAMD	RTNDEVN	MAXELIG	RTNUNAFF
DEVCOUNT	not valid	not valid	not valid	not valid	not valid
DEVLIST	not valid	not valid	not valid	not valid	not valid
DEVSTAT	not valid	not valid	not valid	not valid	not valid
UNITNAME	UNITNAME or DEVTYPE required	not valid	UNITNAME or DEVTYPE required	UNITNAME or DEVTYPE required	not valid

Parameters	RTNATTR	RTNNAMD	RTNDEVN	MAXELIG	RTNUNAFF
DEVTYPE	DEVTYPE or UNITNAME required	not valid	DEVTYPE or UNITNAME required	DEVTYPE or UNITNAME required	not valid
SUBPOOL	not valid	optional	not valid	not valid	not valid
UCBALIST	not valid	not valid	not valid	not valid	not valid
UCBLIST	not valid	not valid	not valid	not valid	not valid
GRIDLIST	not valid	not valid	not valid	not valid	not valid
ATTRAREA	required	not valid	not valid	not valid	not valid
DEVCLASS	not valid	required	not valid	not valid	not valid
NAMELIST	not valid	required	not valid	not valid	not valid
DYNAMIC	not valid	not valid	optional	not valid	not valid
LOC	not valid	not valid	optional	not valid	not valid
RANGE	not valid	not valid	optional	not valid	not valid
DEVNLIST	not valid	not valid	required	not valid	not valid
RECMODE	not valid	not valid	not valid	required	not valid
DENSITY	not valid	not valid	not valid	required	not valid
OUTUNIT	not valid	not valid	not valid	OUTUNIT or OUTDEV required	OUTUNIT or OUTDEV required
OUTDEV	not valid	not valid	not valid	OUTDEV or OUTUNIT required	OUTDEV or OUTUNIT required

Parameters

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

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

Specifies the execute form of the EDTINFO macro.

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

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

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

EDTINFO - Modify form

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

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede EDTINFO.
EDTINFO	
ƀ	One or more blanks must follow EDTINFO.
CHKGRPS	**Note:** At least one of these functions is required: CHKGRPS,
	CHKUNIT, RTNUNIT, RTNUCBA, RTNGRID, RTNATTR,
CHKUNIT	RTNNAMD, RTNDEVN, MAXELIG, RTNUNAFF. If more than
	one of these is specified, a comma must be coded between
RTNUNIT	each of the keywords.
RTNUCBA	**Note:** See the tables following this diagram for information on
	parameter usage with these functions.
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).

Syntax	Description
,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).
,UCBALIST=*ucbalist addr*	*ucbalist addr*: RS-type address or register (2) - (12).
,UCBLIST=*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	**Default**: DYNAMIC=NO
,DYNAMIC=NO	
,LOC=BELOW	**Default**: LOC=BELOW
,LOC=ANY	
,RANGE=ALL	**Default**: RANGE=3DIGIT
,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).

Syntax	Description
,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).
,RSNCODE=*rsncode addr*	*rsncode addr*: RX-type address or register (2) - (12).
,MF=(M,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,MF=(M,*list addr*,COMPLETE)	**Default**: COMPLETE
,MF=(M,*list addr*,NOCHECK)	

The following tables show how the parameters may be specified with the CHKGRPS, 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.

Parameters	CHKGRPS	CHKUNIT	RTNUNIT	RTNUCBA	RTNGRID
DEVCOUNT	required	required	not valid	not valid	not valid
DEVLIST	required	required	not valid	not valid	not valid
DEVSTAT	optional	optional	not valid	not valid	not valid
UNITNAME	not valid	UNITNAME or DEVTYPE required	not valid	UNITNAME or DEVTYPE required	not valid
DEVTYPE	not valid	DEVTYPE or UNITNAME required	required	DEVTYPE or UNITNAME required	not valid
SUBPOOL	not valid	not valid	not valid	optional	not valid
UCBALIST	not valid	not valid	not valid	required	not valid
UCBLIST	not valid	not valid	not valid	not valid	required
GRIDLIST	not valid	not valid	not valid	not valid	required
ATTRAREA	not valid	not valid	not valid	not valid	not valid
DEVCLASS	not valid	not valid	not valid	not valid	not valid
NAMELIST	not valid	not valid	not valid	not valid	not valid
DYNAMIC	not valid	not valid	not valid	not valid	not valid
LOC	not valid	not valid	not valid	not valid	not valid
RANGE	not valid	not valid	not valid	not valid	not valid
DEVNLIST	not valid	not valid	not valid	not valid	not valid

Parameters	CHKGRPS	CHKUNIT	RTNUNIT	RTNUCBA	RTNGRID
RECMODE	not valid	not valid	not valid	not valid	not valid
DENSITY	not valid	not valid	not valid	not valid	not valid
OUTUNIT	not valid	not valid	required	not valid	not valid
OUTDEV	not valid	not valid	not valid	not valid	not valid

Parameters	RTNATTR	RTNNAMD	RTNDEVN	MAXELIG	RTNUNAFF
DEVCOUNT	not valid	not valid	not valid	not valid	not valid
DEVLIST	not valid	not valid	not valid	not valid	not valid
DEVSTAT	not valid	not valid	not valid	not valid	not valid
UNITNAME	UNITNAME or DEVTYPE required	not valid	UNITNAME or DEVTYPE required	UNITNAME or DEVTYPE required	not valid
DEVTYPE	DEVTYPE or UNITNAME required	not valid	DEVTYPE or UNITNAME required	DEVTYPE or UNITNAME required	not valid
SUBPOOL	not valid	optional	not valid	not valid	not valid
UCBALIST	not valid	not valid	not valid	not valid	not valid
UCBLIST	not valid	not valid	not valid	not valid	not valid
GRIDLIST	not valid	not valid	not valid	not valid	not valid
ATTRAREA	required	not valid	not valid	not valid	not valid
DEVCLASS	not valid	required	not valid	not valid	not valid
NAMELIST	not valid	required	not valid	not valid	not valid
DYNAMIC	not valid	not valid	optional	not valid	not valid
LOC	not valid	not valid	optional	not valid	not valid
RANGE	not valid	not valid	optional	not valid	not valid
DEVNLIST	not valid	not valid	required	not valid	not valid
RECMODE	not valid	not valid	not valid	required	not valid
DENSITY	not valid	not valid	not valid	required	not valid
OUTUNIT	not valid	not valid	not valid	OUTUNIT or OUTDEV required	OUTUNIT or OUTDEV required
OUTDEV	not valid	not valid	not valid	OUTDEV or OUTUNIT required	OUTDEV or OUTUNIT required

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 94. 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 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:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key. For the SMC, ECB, TCB, MASID, and MTCB parameters or when the specified *qname* is ADRDFRAG, ADRDSN, ARCENQG, BWODSN, SYSZ*, SYSCTLG, SYSDSN, SYSIEA01, SYSIEECT, SYSIEFSD, SYSIGGV1, SYSIGGV2, SYSPSWRD, SYSVSAM, or SYSVTOC, the authorization must be **one of the following**: • Supervisor state • PSW key 0-7 • APF-authorized.
Dispatchable unit mode:	Task
Cross memory mode:	For LINKAGE=SVC: PASN=HASN=SASN For LINKAGE=SYSTEM: Any PASN, Any HASN, Any SASN For LINKAGE=SYSTEM with SMC=STEP: PASN=HASN, Any SASN

Environmental factor	Requirement
AMODE:	24- or 31- or 64-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Control parameters must be in the primary address space. Except for the TCB, all parameters can reside above 16 megabytes.

Programming requirements

None.

Restrictions

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

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

The caller cannot have an EUT FRR established.

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

Input register information

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

Output register information

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
One of the following:

- If you specify RET=TEST, RET=USE, RET=CHNG, 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.
- Otherwise: Used as a work register by the system.

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

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

Syntax	Description
,*rname length*	*rname length*: symbol, decimal digit, or register (2) - (12). **Default**: assembled length of *rname* **Note**: Code *rname length* if *rname addr* is a register.
,	
,STEP	**Default**: STEP
,SYSTEM	
,SYSTEMS	
)	
,RET=CHNG	**Default**: RET=NONE
,RET=HAVE	
,RET=TEST	
,RET=USE	
,RET=NONE	
,RNL=YES	**Default**: RNL=YES
,RNL=NO	
,RELATED=*value*	*value*: any valid macro keyword specification.
,LINKAGE=SVC	**DEFAULT**: LINKAGE=SVC
,LINKAGE=SYSTEM	

Parameters

The parameters are explained as follows:

(
Specifies the beginning of the resource description.

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

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

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

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

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

You can code a value 1 - 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 overidden. This parameter ensures that the request will be exclusive regardless of the other parameter.

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

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

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

NONE
Control of all the resources is unconditionally requested.

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

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

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

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

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

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

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 1.

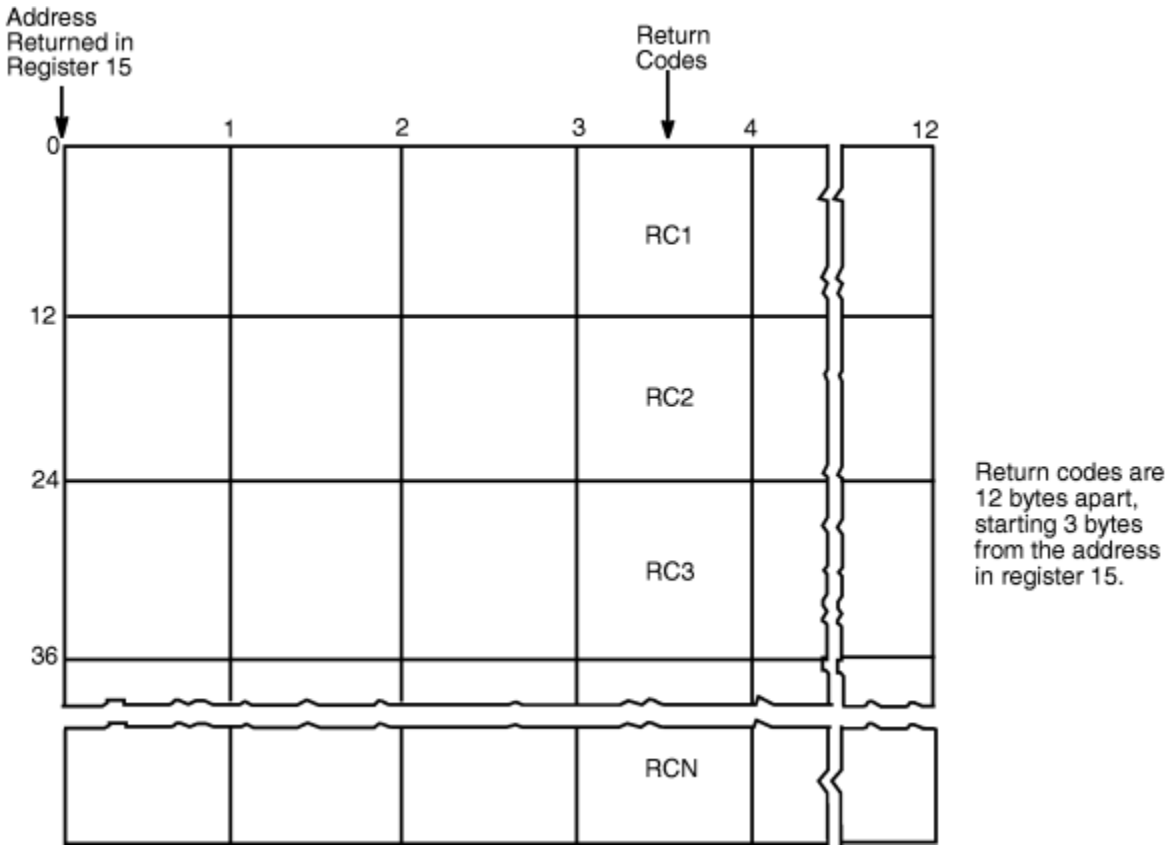


Figure 5. Return Code Area Used by ENQ

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

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

Table 17. Return Codes for the ENQ Macro with the RET=TEST Parameter

Hexadecimal Return Code	Meaning and Action
0	**Meaning**: The resource is immediately available. **Action**: None required. However, you might take some action based on your application.
4	**Meaning**: The resource is not immediately available. **Action**: None required. However, you might take some action based on your application.
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: None required. However, you might take some action based on your application. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.
14	**Meaning**: A previous request for control of the same resource has been made for the same task. The task does not have control of the resource. **Action**: None required. However, you might take some action based on your application.

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

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

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

Table 19. Return Codes for the ENQ Macro with the RET=CHNG Parameter	
Hexadecimal Return Code	**Meaning and Action**
0	**Meaning**: The status of the resource has been changed to exclusive. **Action**: None.
4	**Meaning**: The status of the resource cannot be changed to exclusive. Other tasks share the resource. **Action**: None required. However, you might take some action based on your application.
8	**Meaning**: The status of the resource cannot be changed to exclusive. 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 4.

Table 20. Return Codes for the ENQ Macro with the RET=HAVE Parameter	
Hexadecimal Return Code	**Meaning and Action**
0	**Meaning**: The active task now has control of the resource. **Action**: None.
8	**Meaning**: A previous request for control of the same resource has been made for the same task. The task has control of the resource. **Action**: None required. However, you might take some action based on your application. To determine whether the task has exclusive control or shared control of the resource, check bit 3 of flag byte 1 in the parameter list that identifies the owned resource. If bit 3 is off, the task has exclusive control; If bit 3 is on, the task has shared control.

Table 20. Return Codes for the ENQ Macro with the RET=HAVE Parameter (continued)	
Hexadecimal Return Code	**Meaning and Action**
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 *rname* is allowed to default.

```
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 *rname* length of the first resource is 3 characters and the *rname* length of the third resource is 8 characters. Allow the *rname* length of the second resource to default to its assembled length.

```
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 *qname, rname*, and scope combinations in the list form of the ENQ macro must be equal to the maximum number of *qname, rname*, and scope combinations in any execute form of the macro that refers to that list form.

Syntax

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

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

Syntax	Description
ENQ	
ƀ	One or more blanks must follow ENQ.
(	
qname addr	*qname addr*: A-type address or register (2) - (12).
,	
,rname addr	*rname addr*: A-type address or register (2) - (12).
,	**Default**: E
,E	
,S	
,	
,rname length	*rname length*: symbol or decimal digit.
	Default: assembled length of *rname*
,	**Default**: STEP
,STEP	
,SYSTEM	
,SYSTEMS	
)	
,RET=CHNG	**Default**: RET=NONE
,RET=HAVE	
,RET=TEST	
,RET=USE	
,RET=NONE	
,RNL=YES	**Default**: RNL=YES
,RNL=NO	
,RELATED=*value*	*value*: any valid macro keyword specification.
,MF=L	

Syntax	Description

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:

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

Syntax	Description
,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:

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

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

Chapter 95. 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:

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

Register
Contents

0
Used as a work register by the system

1
Token representing the previously active SPIE/ESPIE environment

2-13
Unchanged

14
Used as a work register by the system

15
Return code of 0

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ESPIE.
ESPIE	
␣	One or more blanks must follow ESPIE.
SET	
,*exit addr*	*exit addr*: A-type address, or register (2) - (12).
,(*interruptions*)	*interruptions*: Decimal 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))
,PARAM=*list addr*	*list addr*: A-type address or register (2) - (12).
,PKM=SYSTEM_RULES	**Default:** PKM=SYSTEM_RULES
,PKM=TOS	

Parameters

The parameters are explained as follows:

SET
Indicates that an ESPIE environment is to be established.

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

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

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

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

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

Note: For 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 for ESPIE—List form

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ESPIE.
ESPIE	
␣	One or more blanks must follow ESPIE.
SET	
,*exit addr*	*exit addr*: A-type address. **Note**: This parameter must be specified on either the list or the execute form of the macro.
,(*interruptions*)	*interruptions*: Decimal digit 1-15 and expressed as:

Syntax	Description
	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 for ESPIE—List form

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 for ESPIE—List form

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 for ESPIE—Execute form

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede ESPIE.
ESPIE	
␣	One or more blanks must follow ESPIE.
SET	

Syntax	Description
,*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	**Default:** PKM=SYSTEM_RULES
,PKM=TOS	

Parameters for ESPIE—Execute form

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 for ESPIE—Execute form

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:

Register
Contents

0
Used as a work register by the system

1
Token identifying the previously active SPIE/ESPIE environment

2-13
Unchanged

14
Used as a work register by the system

15
Return code of 0

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Syntax

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

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

Parameters

The parameters are explained as follows:

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

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

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

Return and reason codes

None.

Example

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

```
ESPIE RESET,TOKEN
```

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:

Register
Contents

0
Used as a work register by the system

1-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Syntax

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

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

Parameters

The parameters are explained as follows:

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

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

Word
Content

0
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)

1
Address of the user-defined parameter list

2
Mask of program interruption types

3
Zero

Return and reason codes

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

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

Hexadecimal Return Code	Meaning
0	**Meaning**: An ESPIE exit is active and the 4-word parameter list contains the information specified in the description of the *parm addr* parameter.
4	**Meaning**: A SPIE exit is active. Word 1 of the parameter list described under *parm addr* contains the address of the current PICA. Words 0, 2, and 3 of the parameter list contain no relevant information.
8	**Meaning**: No SPIE or ESPIE is active. The contents of the 4-word parameter list contain no relevant information.

Example

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

```
ESPIE TEST,PARMLIST
```

Chapter 96. 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.
- 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*.

The descriptions of ESTAE and ESTAEX in this book are:

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

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:

Environmental factor	Requirement
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:

Register
Contents

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

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

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

Syntax	Description
,ASYNCH=NO	
,TERM=NO	**Default**: TERM=NO
,TERM=YES	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,SDWALOC31=NO	**Default**: SDWALOC31=NO
,SDWALOC31=YES	

Parameters

The parameters are explained as follows:

exit addr
0
Specifies the 31-bit address of an ESTAE recovery routine to be entered if the task issuing this macro ends abnormally. If 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.

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

,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.

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

,PURGE=NONE
,PURGE=QUIESCE
,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.

Note:

1. You need to understand PURGE processing before using this parameter. For information about PURGE processing, see *z/OS DFSMSdfp Advanced Services*.
2. When using PURGE, you 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 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.

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.

,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.

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

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

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.

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

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0C	none	**Meaning**: Program error. A recovery routine address equal to zero was specified and either there are no recovery routines for this 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.
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:

Environmental factor	Requirement
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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAEX.
ESTAEX	
ƀ	One or more blanks must follow ESTAEX.
exit addr	*exit addr*: A-type address, or register (2) - (12).
0	
,CT	**Default**: CT

Syntax	Description
,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.

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.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning:** Successful completion of ESTAEX request. **Action:** None.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 parameter area contains data that is not valid. **Action:** Check for an overlay or for use of the execute form without having initialized the list form from a static version of the list form that has the DC for the initial values. Correct the request and either reissue the ESTAEX macro or rerun your program as appropriate.
0C	None	**Meaning:** Program error. A recovery routine address equal to zero was specified and either there are no recovery routines for this TCB. The most recent recovery routine is not owned by the caller, 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.

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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 that 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—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:

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

Syntax	Description
exit addr	*exit addr*: A-type address.
0	
,PARAM=*list addr*	*list addr*: A-type address.
,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.
,SDWALOC31=NO	**Default**: SDWALOC31=NO
,SDWALOC31=YES	**Note:** SDWALOC31 is supported only by ESTAE.
,MF=L	

Parameters

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

,MF=L
Specifies the list form of 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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ESTAE or ESTAEX.
ESTAE ESTAEX	
ƀ	One or more blanks must follow ESTAE or ESTAEX.
exit addr	*exit addr*: RX-type address, or register (2) - (12).
0	
,CT	
,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

Syntax	Description
,SDWALOC31=YES	**Note:** SDWALOC31 is supported only by ESTAE.
,MF=(E,*ctrl addr*)	*ctrl addr*: RX-type address, or register (1) or (2) - (12).

Parameters

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

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

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

Chapter 97. 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 614.

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.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key.
Dispatchable unit mode:	Task
Cross memory mode:	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:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The EVENTS macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede EVENTS.
EVENTS	
␣	One or more blanks must follow EVENTS.
ENTRIES=*n*	*n*: Variable or decimal digit 1-32,767.
ENTRIES=*addr*	*addr*: Register (2) - (12).
ENTRIES=DEL,TABLE=*table*	
address	**Note**: If ENTRIES=*n* or ENTRIES=DEL,TABLE=*table address* is specified, no other parameter should be specified.
TABLE=*table address*	*table address*: Symbol, RX-type address, or register (2) - (12).

Syntax	Description
,WAIT=NO	**Default**: None.
,WAIT=YES	
,ECB=*ecb address*	*ecb address*: Symbol, RX-type address, or register (2) - (12).
,LAST=*last address*	*last address*: Symbol, RX-type address, or register (2) - (12).
	Note: Optional parameters are only valid when TABLE=*table address* is the only required parameter specified.

Parameters

The parameters are explained as follows:

ENTRIES=*n*
ENTRIES=*addr*
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.

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.

Note:

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.

Note:

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.

Note:

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 6 on page 614.

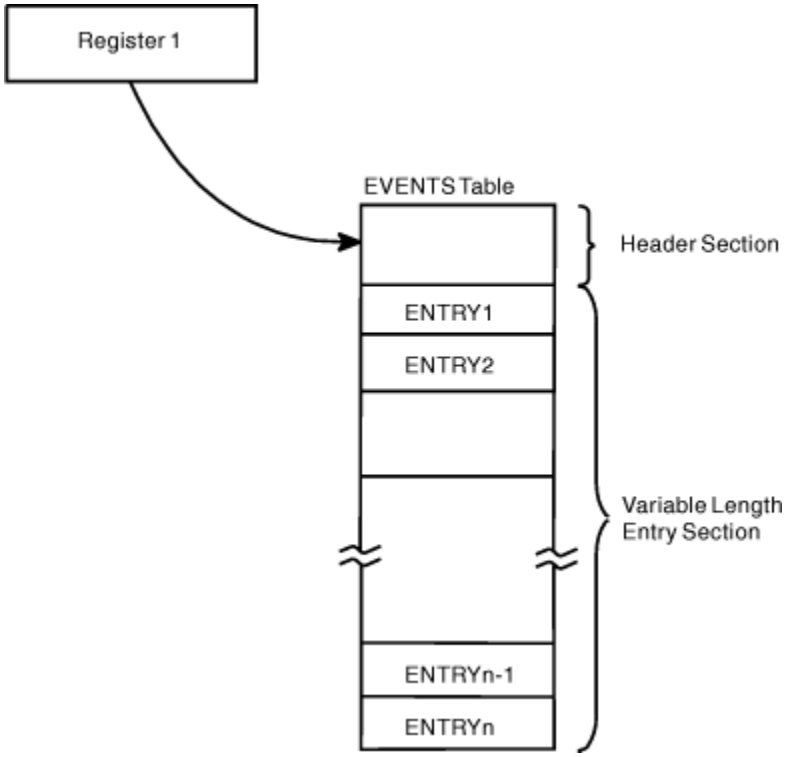


Figure 6. Creating a Table

- 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:

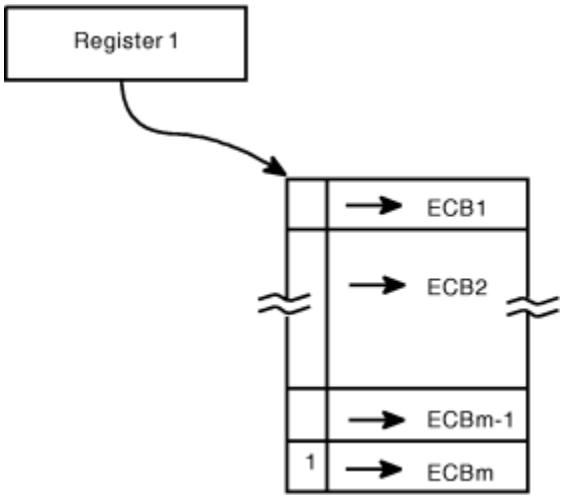


Figure 7. Parameter List Format

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.

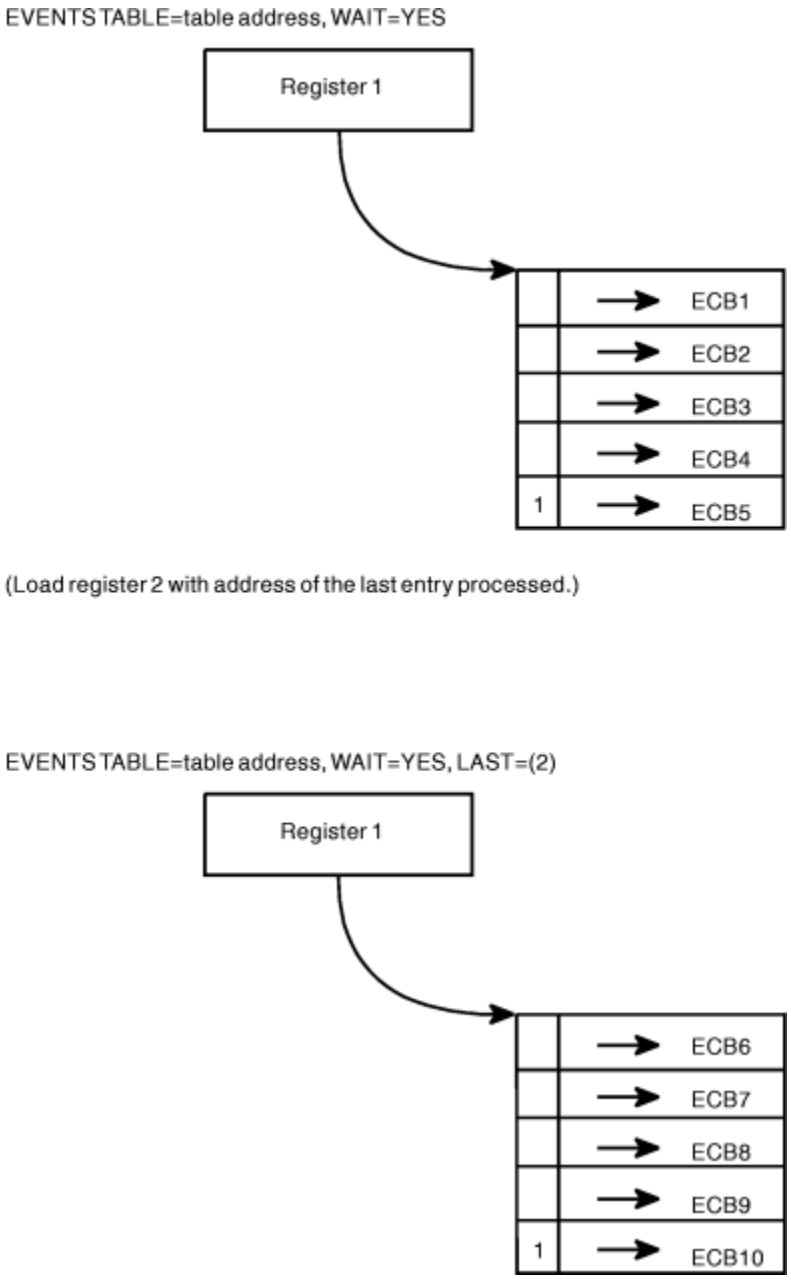


Figure 8. Posting the Parameter List After ECBs 1 through 5 Processed and EVENTS WAIT= Issued

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.

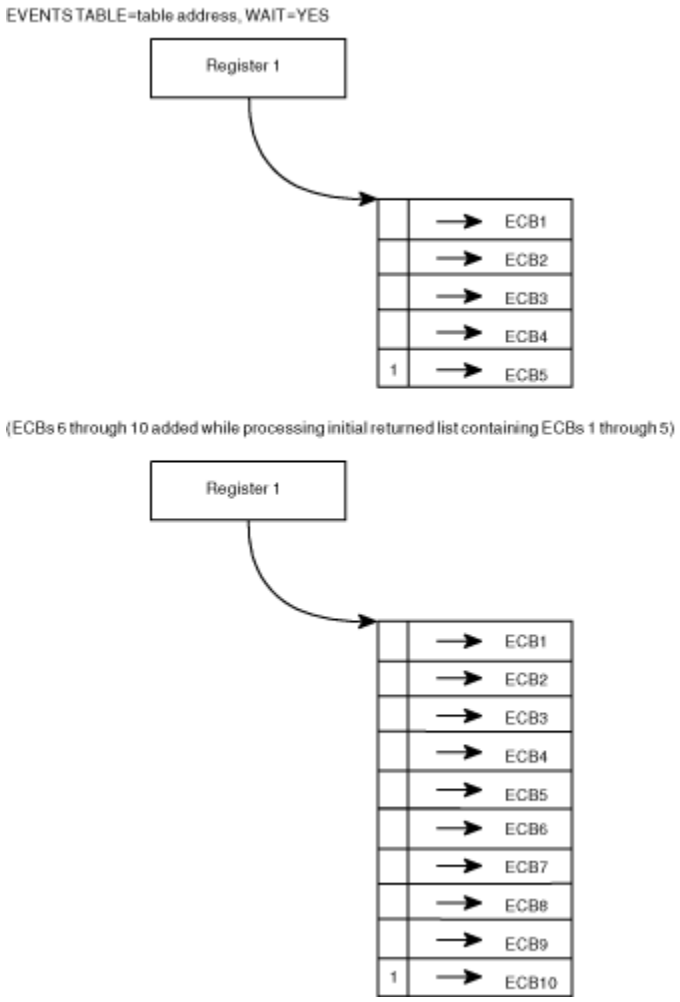


Figure 9. Posting the Parameter List While ECBs 1 through 5 Processed

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'
- 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            CREATE EVENTS TABLE
         ST    R1,TABLE               SAVE EVENTS TABLE
*                                     ADDRESS
NEXTREC  GET   TPDATA,KEY             GET KEY OF NEXT RECORD
*                                     TO PROCESS
         READ  DECBRW,KU,,'S',MF=E    READ THE RECORD
         LA    R3,DECBRW              POINT TO ECB
         EVENTS TABLE=TABLE,ECB=(R3),WAIT=YES  ADD ECB TO
*                                     TABLE AND WAIT UNTIL
*                                     IT IS POSTED
*        PROCESS THE RECORD
         WRITE DECBRW,K,MF=E          WRITE OUT THE RECORD
         LA    R3,DECBRW              POINT TO THE ECB
         EVENTS TABLE=TABLE,ECB=(R3),WAIT=NO
         B     CKRETEST               GO SEE IF IT'S POSTED
RETEST   EVENTS TABLE=TABLE,WAIT=NO   CHECK TO SEE IF ECB IS
*                                     POSTED
CKRETEST LTR   R1,R1                  ANY ECBS POSTED?
         BNZ   NEXTREC                BRANCH IF YES - NEXT
*                                     RECORD
         B     RETEST                 ELSE KEEP CHECKING
TABLE    DS    A                      ADDRESS OF EVENTS TABLE
```

Chapter 98. 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:

Environmental factor	Requirement
Minimum authorization:	For subpools 0-127: problem state and PSW key 8-15. For subpools 131 and 132: a PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be 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.
RMODE:	For SVC-entry, includes 64-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.

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:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
For a conditional request, contains the return code. For an unconditional request, used as a work register by the system.

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

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

Syntax	Description

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FREEMAIN.
FREEMAIN	
ƀ	One or more blanks must follow FREEMAIN.
LC,LA=*length addr*	*length addr*: A-type address, or register (2) - (12).
LU,LA=*length addr*	
L,LA=*length addr*	
VC	
VU	
V	
EC,LV=*length value*	*length value*: symbol, decimal number, or register (2) - (12).
EU,LV=*length value*	
E,LV=*length value*	
RC,LV=*length value*	If R, RC, or RU is specified, register (0) may also be used.
RC,SP=*subpool nmbr*	*subpool nmbr*: symbol, decimal number 0-127, 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.

Syntax	Description
,KEY=*number*	*nmbr*: decimal numbers 0-15, or register (2) - (12).
	Note: KEY may be specified only with RC or RU.
,RELATED=*value*	*value*: any valid assembler character string.

Parameters

The parameters are explained as follows:

LC,LA=*length addr*
LU,LA=*length addr*
L,LA=*length addr*
VC
VU
V
EC,LV=*length value*
EU,LV=*length value*
E,LV=*length value*
RC,LV=*length value*
RC,SP=*subpool nmbr*
RU,LV=*length value*
RU,SP=*subpool nmbr*
R,LV=*length value*
R,SP=*subpool nmbr*

Specifies the type of FREEMAIN request:

LC, LU, and L indicate conditional (LC) and unconditional (LU and L) list requests and specify release of one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas must be provided in a corresponding list whose address is specified in the A parameter. All virtual storage areas must start on a doubleword boundary.

VC, VU, and V indicate conditional (VC) and unconditional (VU and V) variable requests and specify release of single areas of virtual storage. The address and length of the virtual storage area are provided at the address specified in the A parameter.

EC, EU, and E indicate conditional (EC) and unconditional (EU and E) element requests and specify release of single areas of virtual storage. The length of the single virtual storage area is indicated in the LV parameter. The address of the virtual storage area is provided at the address indicated in the A parameter.

RC, RU, and R indicate conditional (RC) and unconditional (RU and R) register requests and specify either the release of all the storage in a subpool or the release of a certain area in a subpool. For information on how to release all the storage in a subpool, see the description for the SP parameter. If the release is for a certain area in a subpool, the address of the virtual storage area is indicated in the A parameter. The length of the area is indicated in the LV parameter. The virtual storage area must start on a doubleword boundary.

Note:

1. For a conditional request, errors detected while processing a FREEMAIN request with incorrect or inconsistent parameters cause the FREEMAIN service to return to the caller with a non-zero return code. For all other errors, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.

For an unconditional request, the system abnormally ends the active task if the FREEMAIN request cannot be successfully completed.

2. If the address of the area to be freed is above 16 megabytes, you must use RC or RU.

LA specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary. One word is required for each virtual storage area to be released; the high-order bit in the last word must be set to 1 to indicate the end of the list. Each word must contain the required length in the low-order three bytes. The fullwords in this list must correspond with the fullwords in the associated list specified in the A parameter. The words must not be in the area to be released. If this rule is violated and if the words are the last allocated items on a virtual page, the whole page is returned to storage and the FREEMAIN abends with an X'0C4' abend code.

LV specifies the length, in bytes, of the virtual storage area being released. The value should be a multiple of 8; if it is not, the control program uses the next high multiple of 8.

- If you specify R,LV=(0) you cannot specify the SP parameter. You must specify the subpool in register 0; the high-order byte must contain the subpool number and the low-order three bytes must contain the length unless you are requesting a subpool release. On a subpool release, the low-order three bytes must contain zeros.
- If you specify R,LV using a symbol, decimal number, or register 2-12, you can specify the SP parameter using registers 0 or 2-12.

,A=*addr*
Specifies the virtual storage address of one or more consecutive fullwords starting on a fullword boundary.

- If E, EC, or EU is coded, one word is required, which contains the address of the virtual storage area to be released.
- If V, VC, or VU is coded, two words are required; the first word contains the address of the virtual storage area to be released, and the second word contains the length of the area to be released.
- If L, LC, or LU is coded, one word is required for each virtual storage area to be released; each word contains the address of one virtual storage area.
- If R, RC, or RU is coded, one word is required, which contains the address of the virtual storage area to be released. If R, RC, or RU is coded and *addr* specifies a register, register 1 through 12 can be used and must contain the address of the virtual storage area to be released.

Do not specify a storage address of 0 with a storage length of 0. This combination causes FREEMAIN to free the subpool specified with the SP parameter, or subpool 0 if the SP parameter is omitted.

,SP=*subpool nmbr*
Specifies the subpool number of the virtual area to be released. Valid subpools numbers are 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=*key number*
Specifies the storage key in which the storage was obtained. The valid storage keys are 0-15. If a register is specified, the storage key must be in bits 24-27 of the register. KEY can be specified for 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*.

- 105
- 10A
- 178
- 205
- 20A
- 278
- 305
- 30A
- 378
- 40A
- 478
- 505
- 605
- 705
- 70A
- 778
- 805
- 80A
- 878
- 905
- 90A
- 978
- A05
- A0A
- A78
- B05
- B0A
- B78
- D05
- D0A
- D78

Return and reason codes

When the FREEMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

Table 21. Return Codes for the FREEMAIN Macro

Return Code	Meaning and Action
0	**Meaning**: Successful completion. **Action**: None.

Table 21. Return Codes for the FREEMAIN Macro (continued)	
Return Code	**Meaning and Action**
4	**Meaning**: Program error. Not all requested virtual storage was freed. **Action**: Check your program for the following kinds of errors: • The address of the storage area to be freed is not correct. • The subpool you have specified does not match the subpool of the storage to be freed. • The key you have specified does not match the key of the storage to be freed.
8	**Meaning**: Program error. No virtual storage was freed because part of the storage area to be freed is fixed. **Action**: Determine whether you have made one of the following errors. If so, correct your program and rerun it: • You passed an incorrect storage area address to the FREEMAIN macro. • You attempted to free storage that is fixed.

Example 1

Free 400 bytes of storage from subpool 10. Register 1 contains the address of the storage area. If the storage is not allocated to the current task, do not abnormally terminate the caller.

```
FREEMAIN RC,LV=400,A=(1),SP=10
```

Example 2

Free all of subpool 3 (if any) that belongs to the current task. If the request is not successful, abnormally terminate the caller.

```
FREEMAIN RU,SP=3
```

Example 3

Free from subpool 5, three areas of storage of 200, 800, and 32 bytes, previously obtained using the list and execute forms of the GETMAIN macro. Storage area addresses are in AREAADD. If any of the storage areas to be freed are not allocated to the current task, abnormally terminate the caller.

```
FREEMAIN LU,LA=LNTHLIST,A=AREAADD,SP=5
  .
  .
  .
LNTHLIST DC F'200',F'800',X'80',FL3'32'
AREAADD  DS 3F
```

FREEMAIN - List form

Use the list form of the FREEMAIN macro to construct a nonexecutable control program parameter list.

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

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

Syntax	Description
ƀ	One or more blanks must follow FREEMAIN.
LC	
LU	
L	
VC	
VU	
V	
EC	
EU	
E	
,LA=*length addr*	*length addr*: A-type address.
,LV=*length value*	*length value*: symbol or decimal number.
	Note: 1. LA may only be specified with LC, LU, or L above. 2. LV may only be specified with EC, EU, or E above.
,A=*addr*	*addr*: A-type address.
,SP=*subpool nmbr*	*subpool nmbr*: symbol or decimal number.
,RELATED=*value*	*value*: any valid assembler character string.
,MF=L	

Parameters

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

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

FREEMAIN - Execute form

A remote control program parameter list is used in, and can be modified by, the execute form of the FREEMAIN macro. The parameter list can be generated by the list form of either a GETMAIN or a FREEMAIN.

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

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede FREEMAIN.
FREEMAIN	
ƀ	One or more blanks must follow FREEMAIN.
LC	
LU	
L	
VC	
VU	
V	
EC	
EU	
E	
,LA=*length addr*	*length addr*: RX-type address or register (2) - (12).
,LV=*length value*	*length value*: symbol, decimal number, or register (2) - (12).
	Note: 1. LA may only be specified with LC, LU, or L above. 2. LV may only be specified with EC, EU, or E above.
,A=*addr*	*addr*: RX-type address, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr*: symbol, decimal number, or register (0) or (2) - (12).
,RELATED=*value*	*value*: any valid assembler character string.

Syntax	Description
,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 99. FXECNTRL — Maintain Function Exploitation and Enablement

Description

The FXECNTRL macro allows fields in the IBM Function Registry for z/OS to be updated.

Additional references and an overview of the IBM Function Registry for z/OS can be found in the *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. PSW key 8-15
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). Control parameters can be above 2G for an AMODE64 caller. The user-provided text strings referenced by the VENDORNAME, PRODUCTNAME, FUNCTIONNAME, PRODUCTID, and INSTANCEID have the same requirements and restrictions as the control parameters.

Programming Requirements

The caller can include the FXEZCTRL macro to get equate symbols for the return and reason codes.

Restrictions

None.

Input Register Information

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

Output Register Information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, when register 15 is not 0.

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

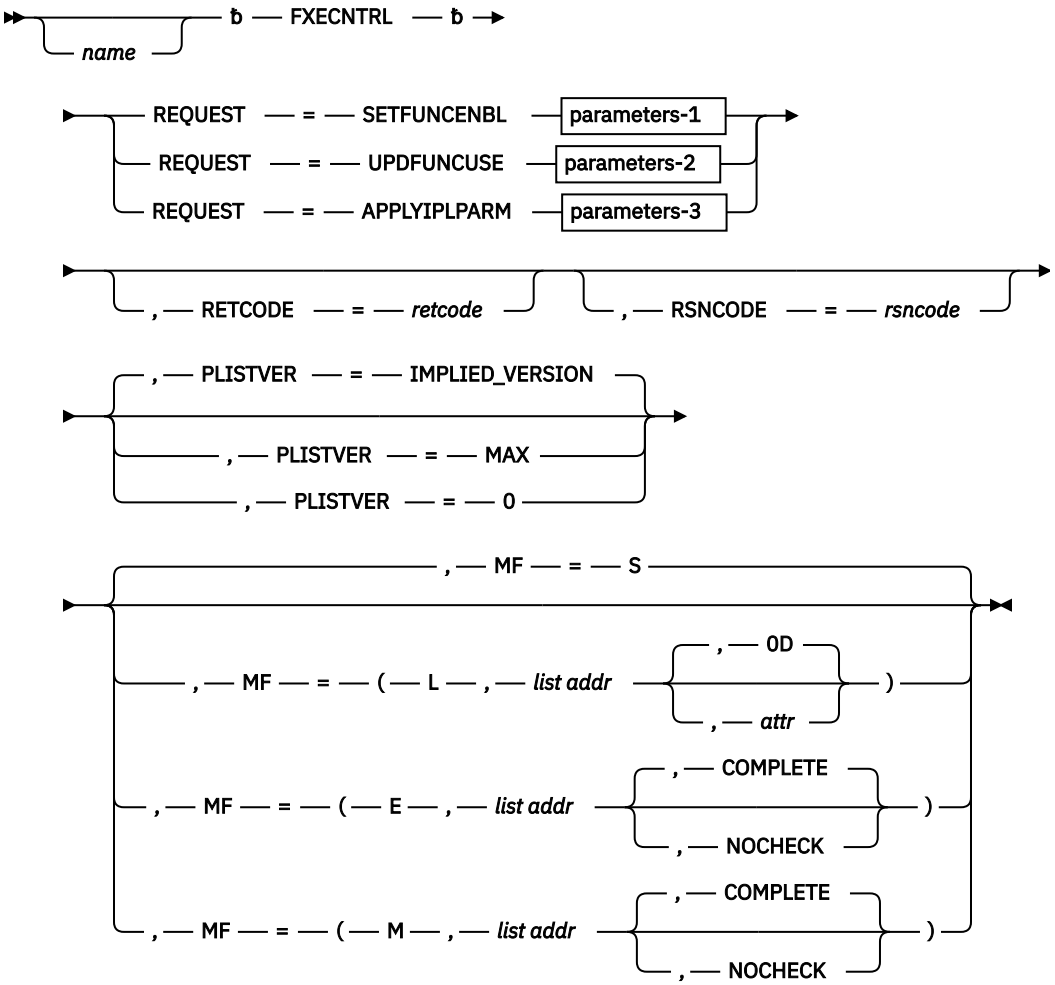
Some callers depend on register contents remaining the same before and after issuing a service. If the system changes the contents of registers on which the caller depends, the caller must save them before issuing the service, and restore them after the system returns control.

Performance Implications

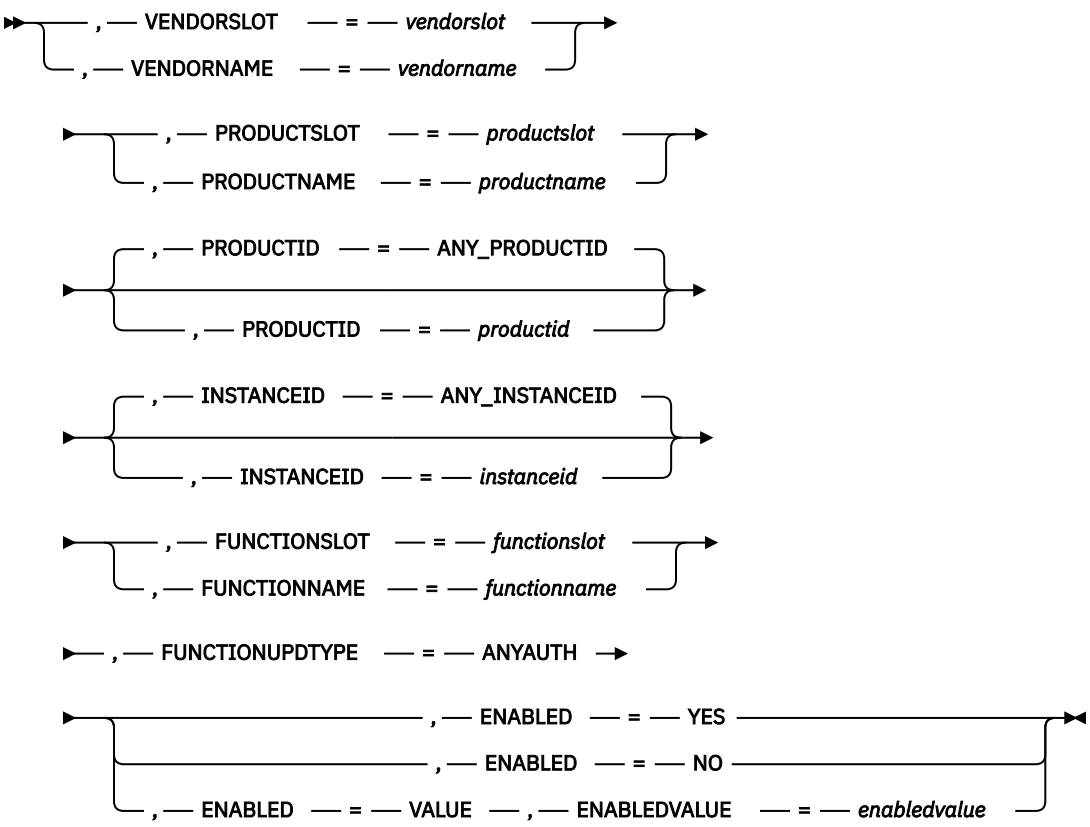
None.

Syntax

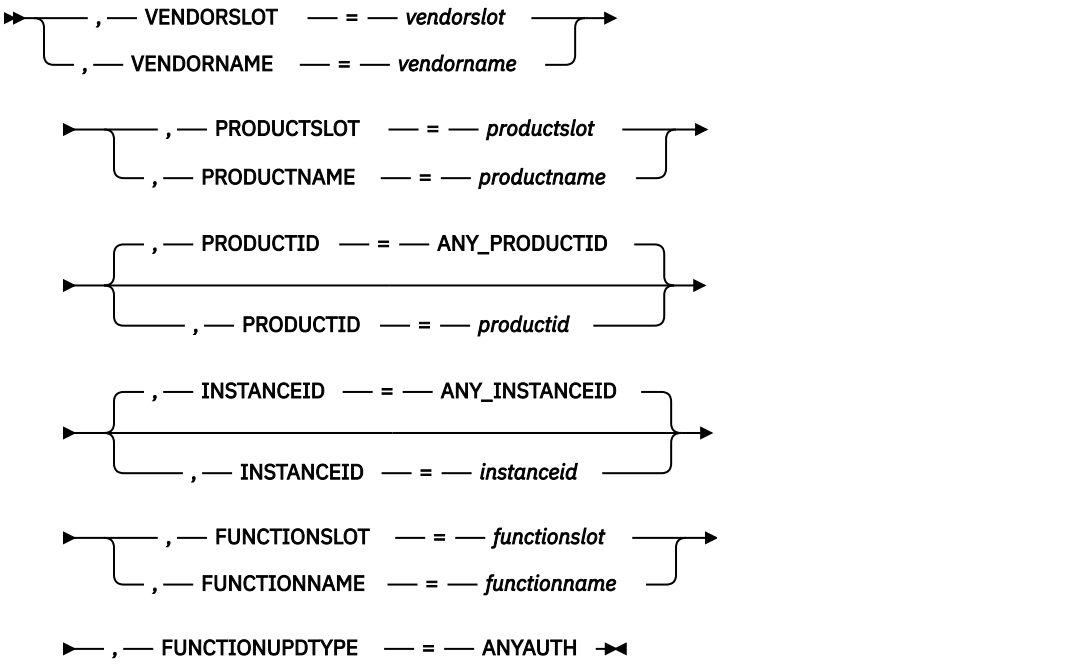
main diagram



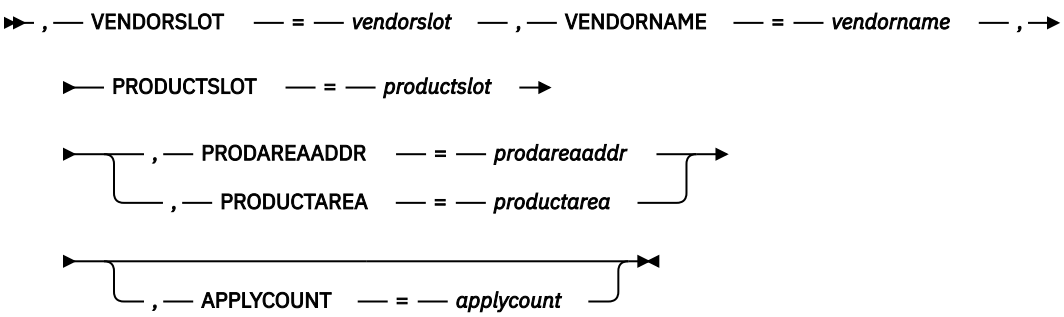
parameters-1



parameters-2



parameters-3



Parameters

The parameters are explained as follows:

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

,APPLYCOUNT=*applycount*
When REQUEST=APPLYIPLPARM is specified, an optional output parameter, field that will contain the count of successful APPLY operations of deferred-apply statements to function entries in the registry for this APPLYIPLPARM request.

Note:

- This is not a count of how many unique function entries were affected by this request, since a single function entry could be a match for more than one statement and the overall APPLYCOUNT is incremented for each of those matching statements.
- A non-zero RETCODE and RSNCODE combination makes the information returned in this field incomplete (for example, unsuccessful APPLY operations for individual function entries are not counted) or even undefined (for example for parameter list access errors or service authorization errors).

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

,ENABLED=YES
,ENABLED=NO
,ENABLED=VALUE
When REQUEST=SETFUNCENBL is specified, a required parameter, which indicates the value to set the selected function's enablement state to.

Note that some function entries might not be set up to have its enablement state changed or tracked at all. In such a case the request will end with a non-zero return code and the enablement state will not be changed.

,ENABLED=YES
indicates to change the enablement state to enabled.

,ENABLED=NO
indicates to change the enablement state to disabled.

,ENABLED=VALUE
indicates that the desired enablement state is specified as runtime value via parameter ENABLEDVALUE.

,ENABLEDVALUE=*enabledvalue*
When ENABLED=VALUE and REQUEST=SETFUNCENBL are specified, a required input parameter, which indicates the desired enablement state:

- specify 1 (one, see also equate *_XENABLED_YES) for enabled.
- specify 0 (zero, see also equate *_XENABLED_NO) for disabled.

To code: Specify the RS-type address of an 8 bit field.

,FUNCTIONNAME=*functionname*
When REQUEST=SETFUNCENBL is specified, a required input parameter which contains the name of the function that is to be used for the requested operation. If FUNCTIONNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,FUNCTIONNAME=*functionname*
When REQUEST=UPDFUNCUSE is specified, a required input parameter which contains the name of the function that is to be used for the requested operation. If FUNCTIONNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,FUNCTIONSLOT=*functionslot*
When REQUEST=SETFUNCENBL is specified, a required input parameter that identifies the slot in the product area that is holding the data for the function entry that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

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

,FUNCTIONSLOT=*functionslot*
When REQUEST=UPDFUNCUSE is specified, a required input parameter that identifies the slot in the product area that is holding the data for the function entry that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

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

,FUNCTIONUPDTYPE=ANYAUTH
When REQUEST=SETFUNCENBL is specified, a required parameter, which indicates in which set of function entries in the selected product area to look for the specified function.

,FUNCTIONUPDTYPE=ANYAUTH
selects the set which authorized and unauthorized callers are allowed to update.

,FUNCTIONUPDTYPE=ANYAUTH
When REQUEST=UPDFUNCUSE is specified, a required parameter, which indicates in which set of function entries in the selected product area to look for the specified function.

,FUNCTIONUPDTYPE=ANYAUTH
selects the set which authorized and unauthorized callers are allowed to update.

,FUNCUPDTYPVALUE=*funcupdtypvalue*
When FUNCTIONUPDTYPE=VALUE and REQUEST=SETFUNCENBL are specified, a required input parameter, which indicates in which set of function entries in the selected product area to look for the specified function:

- specify 0 (zero, see also equate *_XFUNCTIONUPDTYPE_ANYAUTH) to select the set which authorized and unauthorized callers are allowed to update.

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

,FUNCUPDTYPVALUE=*funcupdtypvalue*
When FUNCTIONUPDTYPE=VALUE and REQUEST=UPDFUNCUSE are specified, a required input parameter, which indicates in which set of function entries in the selected product area to look for the specified function:

- specify 0 (zero, see also equate *_XFUNCTIONUPDTYPE_ANYAUTH) to select the set which authorized and unauthorized callers are allowed to update.

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

,INSTANCEID=*instanceid*
,INSTANCEID=ANY_INSTANCEID
When REQUEST=SETFUNCENBL is specified, an optional input parameter that optionally qualifies the desired product further, for example via a subsystem name, especially in the case of multiple instances of the product on the system. If INSTANCEID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_INSTANCEID.

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

,INSTANCEID=*instanceid*
,INSTANCEID=ANY_INSTANCEID
When REQUEST=UPDFUNCUSE is specified, an optional input parameter that optionally qualifies the desired product further, for example via a subsystem name, especially in the case of multiple instances of the product on the system. If INSTANCEID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_INSTANCEID.

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

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
An optional input parameter that specifies the macro form.

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

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

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

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

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

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

,list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1) - (12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

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

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,PRODAREAADDR=*prodareaaddr*
When REQUEST=APPLYIPLPARM is specified, a required input parameter of the product area to apply any matching deferred-apply statements to. IBM recommends to point the FXECNTRL REQUEST=APPLYIPLPARM to a product area just before the area gets linked into the Function Registry infrastructure, not after.

Note:

- Only this one, directly referenced product area is used for this request. Any further product area instances, as optionally linked via the product area's FXEFRPA_NextInstanceAddr field, are not processed automatically and require their own, separate FXECNTRL REQUEST=APPLYIPLPARM calls.
- The product area is expected to reside in common storage.

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

,PRODUCTAREA=*productarea*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that is the product area to apply any matching deferred-apply statements to. IBM recommends to point the FXECNTRL REQUEST=APPLYIPLPARM to a product area just before the area gets linked into the Function Registry infrastructure, not after.

Note:

- Only this one, directly referenced product area is used for this request. Any further product area instances, as optionally linked via the product area's FXEFRPA_NextInstanceAddr field, are not processed automatically and require their own, separate FXECNTRL REQUEST=APPLYIPLPARM calls.
- The product area is expected to reside in common storage.

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

,PRODUCTID=*productid*
,PRODUCTID=ANY_PRODUCTID
When REQUEST=SETFUNCENBL is specified, an optional input parameter that qualifies the desired product further, for example via an FMID, especially in the case of multiple instances of the product on the system. If PRODUCTID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_PRODUCTID.

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

,PRODUCTID=*productid*
,PRODUCTID=ANY_PRODUCTID
When REQUEST=UPDFUNCUSE is specified, an optional input parameter that qualifies the desired product further, for example via an FMID, especially in the case of multiple instances of the product on the system. If PRODUCTID is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

The default is ANY_PRODUCTID.

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

,PRODUCTNAME=*productname*
When REQUEST=SETFUNCENBL is specified, a required input parameter which contains the name of the product that is to be used for the requested operation. If PRODUCTNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,PRODUCTNAME=*productname*
When REQUEST=UPDFUNCUSE is specified, a required input parameter which contains the name of the product that is to be used for the requested operation. If PRODUCTNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

To code: Specify the RS-type address, or address in register (2) - (12), of a 48-character field.

,PRODUCTSLOT=*productslot*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that identifies the product slot, in the specified vendor area, which will be, or already is, holding the pointer to the specified product area. Slot numbers are 1-based, meaning the first slot is slot number 1.

You must specify both PRODUCTSLOT and the PRODAREAADDR or PRODUCTAREA parameter (which implies the product name, instance ID and product ID information) to ensure proper matching against either product slot or product name syntax, as chosen by the user in the deferred-apply statements from the FXEPRMxx PARMLIB members identified by the system parameter FXE.

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

,PRODUCTSLOT=*productslot*
When REQUEST=SETFUNCENBL is specified, a required input parameter that identifies the slot in the selected vendor area that is holding the pointer to the product area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

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

,PRODUCTSLOT=*productslot*
When REQUEST=UPDFUNCUSE is specified, a required input parameter that identifies the slot in the selected vendor area that is holding the pointer to the product area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

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

REQUEST=APPLYIPLPARM
REQUEST=SETFUNCENBL
REQUEST=UPDFUNCUSE
A required parameter, which identifies the type of request.

REQUEST=SETFUNCENBL
Set the enablement state for a function of a given product, owned by a given vendor. Wildcard use in the selectors is allowed and FXECNTRL will attempt to update all matching function entries.

REQUEST=UPDFUNCUSE
Update the usage information for a function of a given product, owned by a given vendor, based on the current function usage indicator field FXEFRFE_Used of the selected function entry:

- If FXEFRFE_Used=FXEFRFE_Used_NO_Num, then field FXEFRFE_Used is set to FXEFRFE_Used_YES_Num, indicating that the function has been used at least once.
- If FXEFRFE_Used=FXEFRFE_Used_YES_Num, field FXEFRFE_Used will not be changed and it will continue to indicate that the function has been used at least once.
- If FXEFRFE_Used=FXEFRFE_Used_USE_OTHER_Num, then field FXEFRFE_UsageCount will be incremented by one.
- If FXEFRFE_Used=FXEFRFE_Used_NOT_TRACKED_Num, then the request ends with a warning since the function owner set this function up to not have any usage tracked. This function entry will not be updated.

Wildcard use in the selectors is allowed and FXECNTRL will attempt to update all matching function entries.

REQUEST=APPLYIPLPARM
Apply any matching deferred-apply statements to the function entries of the single product area specified by the PRODUDTAREA or PRODAREAADDR parameter. These statements are from FXEPRMxx PARMLIB members that were identified by the system parameter FXE at system start time, but which could not be applied at that time yet, since the Function Registry infrastructure had not been fully established. IBM recommends to call FXECNTRL REQUEST=APPLYIPLPARM for newly created product areas just before such a product area gets added to the Function Registry infrastructure to ensure that the product area is properly primed, before any end user, or other

product code, can start modifying contained function entries via interfaces like the SETFXE command, the SET FXE command, the FXECNTRL service or via direct memory access.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2) - (12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2) - (12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,VENDORNAME=*vendorname*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that contains the vendor name from the vendor area which will be, or already is, referencing the specified product area.

Both VENDORNAME and VENDORSLOT parameters have to be specified to ensure proper matching against either vendor name or vendor slot syntax, as chosen by the user in the deferred-apply statements from the FXEPRMxx PARMLIB members identified by the system parameter FXE.

The VENDORNAME value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

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

,VENDORNAME=*vendorname*
When REQUEST=SETFUNCENBL is specified, a required input parameter which contains the name of the vendor that is to be used for the requested operation. If VENDORNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

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

,VENDORNAME=*vendorname*
When REQUEST=UPDFUNCUSE is specified, a required input parameter which contains the name of the vendor that is to be used for the requested operation. If VENDORNAME is specified, the value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.

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

,VENDORSLOT=*vendorslot*
When REQUEST=APPLYIPLPARM is specified, a required input parameter that identifies the slot in the anchor table that will be, or already is, holding the pointer to the vendor area which will be referencing the specified product area. Slot numbers are 1-based, meaning the first slot is slot number 1.

Both VENDORSLOT and VENDORNAME parameters have to be specified to ensure proper matching against either vendor slot or vendor name syntax, as chosen by the user in the deferred-apply statements from the FXEPRMxx PARMLIB members identified by the system parameter FXE.

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

,VENDORSLOT=*vendorslot*
When REQUEST=SETFUNCENBL is specified, a required input parameter that identifies the slot in the anchor table that is holding the pointer to the vendor area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

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

,VENDORSLOT=*vendorslot*
When REQUEST=UPDFUNCUSE is specified, a required input parameter that identifies the slot in the anchor table that is holding the pointer to the vendor area that is to be used for the requested operation. Slot numbers are 1-based, meaning the first slot is slot number 1.

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

ABEND Codes

None.

Return and Reason codes

When the FXECNTRL macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro FXEZCTRL provides equate symbols for the return and reason codes.

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

Table 22. Return and reason codes for the FXECNTRL macro

Return code	Reason code	Equate symbol, meaning and action
0	–	**Equate symbol**: FxeCntrlRc_OK **Meaning**: Successfully executed the request. **Action**: None required
4	–	**Equate symbol**: FxeCntrlRc_Warning **Meaning**: Warning **Action**: Refer to action under the individual reason code.
4	*xxxx*04F1	**Equate symbol**: FxeCntrlRsn_VendorNotFound **Meaning**: Requested function entry not found: No vendor matched. **Action**: Ensure that the given vendor, product, and function selectors identify an existent entry in the IBM Function Registry for z/OS.
4	*xxxx*04F2	**Equate symbol**: FxeCntrlRsn_ProductNotFound **Meaning**: Requested function entry not found: None of the matching vendors had a matching product. **Action**: Ensure that the given vendor, product, and function selectors identify an existent entry in the IBM Function Registry for z/OS.

Table 22. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
4	*xxxx*04F3	**Equate symbol**: FxeCntrlRsn_FunctionNotFound **Meaning**: Requested function entry not found: None of the matching products had a matching function entry. **Action**: Ensure that the given vendor, product, and function selectors identify an existent entry in the IBM Function Registry for z/OS.
8	–	**Equate symbol**: FxeCntrlRc_Error **Meaning**: Error **Action**: Refer to action under the individual reason code.
8	*xxxx*0801	**Equate symbol**: FxeCntrlRsn_BadEnvNotEnabled **Meaning**: Not enabled. **Action**: Avoid using FXECNTRL when not enabled for I/O and external interrupts.
8	*xxxx*0802	**Equate symbol**: FxeCntrlRsn_BadEnvLocked **Meaning**: Locked. **Action**: Avoid using FXECNTRL when a lock is held.
8	*xxxx*0803	**Equate symbol**: FxeCntrlRsn_BadEnvSrbmode **Meaning**: SRB mode. **Action**: Avoid issuing FXECNTRL in SRB mode.
8	*xxxx*0804	**Equate symbol**: FxeCntrlRsn_BadEnvFRR **Meaning**: The caller had an EUT FRR established. **Action**: Avoid using FXECNTRL when an EUT FRR is established.
8	*xxxx*0805	**Equate symbol**: FxeCntrlRsn_BadParmlist **Meaning**: Error accessing parameter list. **Action**: Make sure that the provided parameter list is valid.
8	*xxxx*0806	**Equate symbol**: FxeCntrlRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure that the ALET associated with the parameter list is valid. The access register might not have been set up correctly.
8	*xxxx*0807	**Equate symbol**: FxeCntrlRsn_BadVendorName **Meaning**: Error accessing VENDORNAME. **Action**: Make sure that the provided VENDORNAME parameter is valid.
8	*xxxx*0808	**Equate symbol**: FxeCntrlRsn_BadVendorNameALET **Meaning**: Bad VENDORNAME ALET. **Action**: Make sure that the ALET associated with the VENDORNAME parameter is valid. The access register might not have been set up correctly.

Table 22. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	*xxxx*0809	**Equate symbol**: FxeCntrlRsn_BadProductName **Meaning**: Error accessing PRODUCTNAME. **Action**: Make sure that the provided PRODUCTNAME parameter is valid.
8	*xxxx*080A	**Equate symbol**: FxeCntrlRsn_BadProductNameALET **Meaning**: Bad PRODUCTNAME ALET. **Action**: Make sure that the ALET associated with the PRODUCTNAME parameter is valid. The access register might not have been set up correctly.
8	*xxxx*080B	**Equate symbol**: FxeCntrlRsn_BadProductID **Meaning**: Error accessing PRODUCTID. **Action**: Make sure that the provided PRODUCTID parameter is valid.
8	*xxxx*080C	**Equate symbol**: FxeCntrlRsn_BadProductIdALET **Meaning**: Bad PRODUCTID ALET. **Action**: Make sure that the ALET associated with the PRODUCTID parameter is valid. The access register might not have been set up correctly.
8	*xxxx*080D	**Equate symbol**: FxeCntrlRsn_BadInstanceID **Meaning**: Error accessing INSTANCEID. **Action**: Make sure that the provided INSTANCEID parameter is valid.
8	*xxxx*080E	**Equate symbol**: FxeCntrlRsn_BadInstanceIdALET **Meaning**: Bad INSTANCEID ALET. **Action**: Make sure that the ALET associated with the INSTANCEID parameter is valid. The access register might not have been set up correctly.
8	*xxxx*080F	**Equate symbol**: FxeCntrlRsn_BadFunctionName **Meaning**: Error accessing FUNCTIONAME. **Action**: Make sure that the provided FUNCTIONAME parameter is valid.
8	*xxxx*0810	**Equate symbol**: FxeCntrlRsn_BadFunctionNameALET **Meaning**: Bad FUNCTIONNAME ALET. **Action**: Make sure that the ALET associated with the FUNCTIONNAME parameter is valid. The access register might not have been set up correctly.
8	*xxxx*0811	**Equate symbol**: FxeCntrlRsn_BadRequest **Meaning**: A bad REQUEST type has been specified. **Action**: Use one of the supported request types.

Table 22. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	*xxxx*0812	**Equate symbol**: FxeCntrlRsn_BadParmlistVersion **Meaning**: The specified version of the macro is not compatible with the current version of IBM Function Registry for z/OS. **Action**: Avoid requesting parameters that are not supported by this version of IBM Function Registry for z/OS.
8	*xxxx*0813	**Equate symbol**: FxeCntrlRsn_BadFunctionUpdateType **Meaning**: Bad function update type. **Action**: Use only supported updated types: ANYAUTH, AUTHONLY, or VALUE. If VALUE, then FUNCUPDTYPVALUE has to be 0 (ANYAUTH) or 1 (AUTHONLY).
8	*xxxx*0814	**Equate symbol**: FxeCntrlRsn_NotAuthorizedForFuncUpdType **Meaning**: Not authorized for FUNCTIONUPDTYPE. **Action**: Only authorized callers are allowed to use FUNCTIONUPDTYPE=AUTHONLY or FUNCTIONUPDTYPE=VALUE and FUNCUPDTYPVALUE=1 (AUTHONLY).
8	*xxxx*0815	**Equate symbol**: FxeCntrlRsn_BadEnabledValue **Meaning**: Bad ENABLED value. **Action**: Use only supported ENABLED values: YES, NO, or VALUE. If VALUE, then ENABLEDVALUE has to be 1 (YES) or 0 (NO).
8	*xxxx*0816	**Equate symbol**: FxeCntrlRsn_EnabledNotTracked **Meaning**: Enablement state is not tracked. The request was aborted and no matching function entries had their enablement state changed. If wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all matching function entries had their enablement state changed. **Action**: The selected function entry is set up to not track enablement state. Refer to the documentation for this function entry for more information about its capabilities.
8	*xxxx*0817	**Equate symbol**: FxeCntrlRsn_UsageNotTracked **Meaning**: Usage information is not tracked. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their usage information changed. **Action**: The selected function entry is set up to not track usage information. Refer to the documentation for this function entry for more information about its capabilities.
8	*xxxx*0818	**Equate symbol**: FxeCntrlRsn_BadVendorSlot **Meaning**: Invalid VENDORSLOT. **Action**: Ensure a valid VENDORSLOT number is specified. In particular 0 (zero) is not valid. Slot numbers start at 1 (one).

Table 22. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	*xxxx*0819	**Equate symbol**: FxeCntrlRsn_BadProductSlot **Meaning**: Invalid PRODUCTSLOT. **Action**: Ensure a valid PRODUCTSLOT number is specified. In particular 0 (zero) is not valid. Slot numbers start at 1 (one).
8	*xxxx*081A	**Equate symbol**: FxeCntrlRsn_BadFunctionSlot **Meaning**: Invalid FUNCTIONSLOT. **Action**: Ensure a valid FUNCTIONSLOT number is specified. In particular 0 (zero) is not valid. Slot numbers start at 1 (one).
8	*xxxx*081B	**Equate symbol**: FxeCntrlRsn_NotAuthorizedForApplyIplParm **Meaning**: Not authorized for REQUEST=APPLYIPLPARM. **Action**: Only authorized callers are allowed to use request type APPLYIPLPARM.
8	*xxxx*081C	**Equate symbol**: FxeCntrlRsn_BadProductArea **Meaning**: Error accessing product area. **Action**:Make sure that the provided product area, referenced by parameter PRODUCTAREA or PRODAREAADDR, is valid (in common storage, writable, not NULL for example).
8	*xxxx*081D	**Equate symbol**: FxeCntrlRsn_BadParmlistUpd **Meaning**: Error updating parameter list for output parameters. **Action**: Make sure that the provided parameter list is valid and writable.
0C	–	**Equate symbol**: FxeCntrlRc_SevereError **Meaning**: Severe Error / Environment Error **Action**: Refer to action under the individual reason code.
0C	*xxxx*0C80	**Equate symbol**: FxeCntrlRsn_RegistryNotAvailable **Meaning**: IBM Function Registry for z/OS is not available. **Action**: This might be a temporary situation in the very early phases of the system start. Contact IBM support if this problem persists and if problem reporting databases do not list an existing fix.
0C	*xxxx*0C81	**Equate symbol**: FxeCntrlRsn_EnabledValNotSupported **Meaning**: Unsupported ENABLED value in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their enablement state changed. **Action**: Contact the owner of the selected function entry. An unsupported enabled value was found in the registry for this function entry.

Table 22. Return and reason codes for the FXECNTRL macro (continued)

Return code	Reason code	Equate symbol, meaning and action
0C	*xxxx*0C82	**Equate symbol**: FxeCntrlRsn_EnabledValLocked **Meaning**: ENABLED locked in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their enablement state changed. **Action**: The selected function entry is not set up to have its enablement state changed. Refer to the documentation for this function entry for more information about its capabilities.
0C	*xxxx*0C83	**Equate symbol**: FxeCntrlRsn_UsageValNotSupported **Meaning**: Unsupported usage indicator in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their usage indicator changed. **Action**: Contact the owner of the selected function entry. An unsupported usage indicator was found in the registry for this function entry.
0C	*xxxx*0C8A	**Equate symbol**: FxeCntrlRsn_EnabledInvalidFuncEntry **Meaning**: Inconsistent function entry (FXEFRFE) in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their enablement state changed. **Action**: Contact the owner of the selected function entry. An inconsistent function entry was found in the registry. For example, the function entry did not have a recognized acronym value set in its FXEFRFE_ID field.
0C	*xxxx*0C8B	**Equate symbol**: FxeCntrlRsn_UsageInvalidFuncEntry **Meaning**: Inconsistent function entry (FXEFRFE) in registry. The request was aborted and no, or, if wildcards have been used in the VENDOR, PRODUCT, or FUNCTION selectors, not all, matching function entries had their function usage indicator changed. **Action**: Contact the owner of the selected function entry. An inconsistent function entry was found in the registry. For example, the function entry did not have a recognized acronym value set in its FXEFRFE_ID field.
10	–	**Equate symbol**: FxeCntrlRc_CompError **Meaning**: Component error. **Action**: Report the associated reason code to the system programmer to contact IBM Service.

Chapter 100. 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.

Note:

1. When you obtain storage, the system clears the requested storage to zeros if you obtain either:
   - 8192 bytes or more from a pageable, private storage subpool.
   - 4096 bytes or more from a pageable, private storage subpool, with BNDRY=PAGE or STARTBDY=12 specified.

     In all other cases you must not assume that the storage is cleared to zeros.

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

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 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 .

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For subpools 0-127: problem state and PSW key 8-15. For subpools 131 and 132: 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.
RMODE:	For SVC-entry, includes 64-bit
ASC mode:	Primary.
Interrupt status:	Enabled for I/O and external interrupts.
Locks:	No locks held.

Environmental factor	Requirement
Control parameters:	For LC, LU, VC, VU, EC, EU requests: control parameters must be in the primary address space. For other requests: control parameters are in registers.

Programming requirements

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:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

For RC, RU, and R requests: when control returns to the caller the GPRs contain:

Register
Contents

0
Used as a work register by the system.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

For VRC and VRU requests: when control returns to the caller the GPRs contain:

Register
Contents

0
For a successful request, contains the length of the storage obtained. Otherwise, used as a work register by the system.

1
The address of the allocated storage when GETMAIN is successful; otherwise, used as a work register by the system.

Note: A successful GETMAIN will return a 64-bit pointer to the obtained area (bits 0-32 will be zero).

2-13
Unchanged.

14
Used as a work register by the system.

15
Contains the return code.

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

Repeatedly issuing the GETMAIN macro can slow down performance. If your program requires many identically sized storage areas, use the CPOOL macro or callable cell pool services for better performance.

Syntax

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

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GETMAIN.
GETMAIN	
ƀ	One or more blanks must follow GETMAIN.
LC,LA=*length addr*,A=*addr*	*length addr*: A-type address, or register (2) - (12).
LU,LA=*length addr*,A=*addr*	*length value*: symbol, decimal number, or register (2) - (12).

Syntax	Description
VC,LA=*length addr*,A=*addr*	If RC or RU is specified, register (0) may also be specified.
VU,LA=*length addr*,A=*addr*	
EC,LV=*length value*,A=*addr*	*addr*: A-type address or register (2) - (12).
EU,LV=*length value*,A=*addr*	**Note:** RC, RU, VRC, or VRU must be used for address greater than 16 megabytes.
RC,LV=*length value*	
RU,LV=*length value*	
R,LV=*length value*	
VRC,LV=*(maximum length value, minimum length value)*	*maximum length value*: symbol, decimal number, or register (2) - (12).
VRU,LV=*(maximum length value, minimum length value)*	*minimum length value*: symbol, decimal number, or register (2) - (12).
,SP=*subpool nmbr*	*subpool nmbr*: symbol or decimal number 0-127, 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	**Note:** This parameter may not be specified with R above.
,CONTBDY=*containing_bdy*	*containing_bdy*: Decimal number 3-31 or register (2) - (12). **Note**: CONTBDY may be specified only with RC or RU.
,STARTBDY=*starting_bdy*	*starting_bdy*: Decimal number 3-31 or register (2) - (12). **Note**: STARTBDY may be specified only with RC or RU.
,KEY=*key number*	*key number*: decimal numbers 0-15, or register (2) - (12).
	Note: KEY may be specified only with RC, RU, VRC, or VRU.
,LOC=24	**Note**: The LOC parameter applies only when used with RC, RU, VRC, or VRU request types. For all other types, LOC=24 is used.
,LOC=(24,31)	
,LOC=(24,64)	

Syntax	Description
,LOC=31	
,LOC=(31,31)	
,LOC=(31,64)	
,LOC=RES	**Default**: LOC=RES (for request types where LOC applies); LOC=24 (for request types where LOC does not apply)
,LOC=(RES,31)	
,LOC=(RES,64)	
,LOC=EXPLICIT	**Note**: You must specify the INADDR parameter with EXPLICIT.
,LOC=(EXPLICIT,24)	
,LOC=(EXPLICIT,31)	
,LOC=(EXPLICIT,64)	
,INADDR=*stor addr*	*stor addr*: RX-type address or register (1)-(12). **Note**: This parameter can only be specified with LOC=EXPLICIT.
,CHECKZERO=YES	**Default:** CHECKZERO=NO
,CHECKZERO=NO	**Note**: CHECKZERO may be specified only with RC, RU, VRC, or VRU.
,RELATED=*value*	*value:* Any valid assembler character string

Parameters

The parameters are explained as follows.

The first parameter of the GETMAIN macro is positional and is required. This parameter describes the type or mode of the GETMAIN request. The first parameter can be one of the following values:

LC,LA=*length addr*, A=*addr*
LU,LA=*length addr*, A=*addr*
VC,LA=*length addr*, A=*addr*
VU,LA=*length addr*, A=*addr*
EC,LV=*length value*, A=*addr*
EU,LV=*length value*, A=*addr*
RC,LV=*length value*
RU,LV=*length value*
R,LV=*length value*
VRC,LV=*(maximum length value,minimum length value)*
VRU,LV=*(maximum length value,minimum length value)*

LC and LU indicate conditional (LC) and unconditional (LU) list requests, and specify requests for one or more areas of virtual storage. The length of each virtual storage area is indicated by the values in a list beginning at the address specified in the LA parameter. The address of each of the virtual storage areas is returned in a list beginning at the address specified in the A parameter. No virtual storage is allocated unless all of the requests in the list can be satisfied.

VC and VU indicate conditional (VC) and unconditional (VU) variable requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is between the two values at the address specified in the LA parameter. The address and actual length of the allocated virtual storage area are returned by the system at the address indicated in the A parameter.

EC and EU indicate conditional (EC) and unconditional (EU) element requests, and specify requests for single areas of virtual storage. The length of the single virtual storage area is indicated by the parameter, LV=*length value*. The address of the allocated virtual storage area is returned at the address indicated in the A parameter.

RU and R indicate unconditional register requests; RC indicates a conditional register request. RC, RU, and R specify requests for single areas of virtual storage. The length of the single virtual area is indicated by the parameter, LV=*length value*. The address of the allocated virtual storage area is returned in register 1.

VRC and VRU indicate variable register conditional (VRC) and unconditional (VRU) requests for a single area of virtual storage. The length returned will be between the maximum and minimum lengths specified by the parameter LV=(*maximum length value, minimum length value*). The address of the allocated virtual storage is returned in register 1 and the length in register 0.

Note:

1. A **conditional request** indicates that the active unit of work is not to be abnormally terminated if there is insufficient contiguous virtual storage to satisfy the request. A conditional request does not prevent all abnormal terminations. For example, if the request has incorrect or inconsistent parameters, the system abnormally terminates the active unit of work. An **unconditional request** indicates that the active unit of work is to be abnormally terminated whenever the request cannot complete successfully.
2. The LC, LU, VC, VU, EC, EU, and R requests can be used only to obtain virtual storage with addresses below 16 megabytes. The RC, RU, VRC, and VRU requests can be used to obtain virtual storage with addresses above 16 megabytes.

LA specifies the virtual storage address of consecutive fullwords starting on a fullword boundary. Each fullword must contain the required length in the low-order three bytes, with the high-order byte set to 0. The lengths should be multiples of 8; if they are not, the system uses the next higher multiple of 8. If VC or VU was coded, two words are required. The first word contains the minimum length required, the second word contains the maximum length. If LC or LU was coded, one word is required for each virtual storage area requested; the high-order bit of the last word must be set to 1 to indicate the end of the list. The list must not overlap the virtual storage area specified in the A parameter.

LV=*length value* specifies the length, in bytes, of the requested virtual storage. The number should be a multiple of 8; if it is not, the system uses the next higher multiple of 8. If R is specified, LV=(0) may be coded; the low-order three bytes of register 0 must contain the length, and the high-order byte must contain the subpool number. LV=(*maximum length value, minimum length value*) specifies the maximum and minimum values of the length of the storage request.

The A parameter specifies the virtual storage address of consecutive fullwords, starting on a fullword boundary. The system places the address of the virtual storage area allocated in one or more words. If E was coded, one word is required. If LC or LU was coded, one word is required for each entry in the LA list. If VC or VU was coded, two words are required. The first word contains the address of the virtual storage area, and the second word contains the length actually allocated. The list must not overlap the virtual storage area specified in the LA parameter.

,SP=*subpool nmbr*

Specifies the number of the subpool from which the virtual storage area is to be allocated. If you specify a register, the subpool number must be in bits 24-31 of the register, with bits 0-23 set to zero. Valid subpool numbers are 0-127, 131, and 132. See the virtual storage management chapter in *z/OS MVS Programming: Assembler Services Guide* for complete information about these subpools.

,BNDRY=DBLWD
,BNDRY=PAGE
Specifies that alignment on a doubleword boundary (DBLWD) or alignment with the start of a virtual page on a 4K boundary (PAGE) is required for the start of a requested area.

,CONTBDY=*containing_bdy*
Specifies the boundary the obtained storage must be contained within. Specify a power of 2 that represents the containing boundary. Supported values are 3-31. For example, CONTBDY=10 means the containing boundary is 2**10, or 1024 bytes. The containing boundary must be at least as large as the maximum requested boundary. The obtained storage will not cross an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24 - 31 of the register. CONTBDY is valid only with RC or RU.

CONTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

CONTBDY applies to all subpools.

Generally, if you omit this parameter, there is no containing boundary. However, if the GETMAIN is for SQA or LSQA, and is for less than 4 KB, and STARTBDY is specified, the default of CONTBDY is 12, ensuring that the GETMAIN stays within a 4 KB page boundary.

For GETMAIN macros that specify a CONTBDY parameter value that is larger than 12, it is possible that the allocated area spans across a 4 KB page boundary, even when the area is less than or equal to 4 KB and in an SQA or LSQA subpool.

,STARTBDY=*starting_bdy*
Specifies the boundary the obtained storage must start on. Specify a power of 2 that represents the start boundary. Supported values are 3-31. For example, STARTBDY=10 means the start boundary is 2**10, or 1024 bytes. The obtained storage will begin on an address that is a multiple of the requested boundary.

If a register is specified, the value must be in bits 24-31 of the register. STARTBDY is valid only with RC or RU.

STARTBDY is not valid with LOC=EXPLICIT or BNDRY=PAGE.

STARTBDY applies to all subpools.

If you omit this parameter, the start boundary is 8 bytes (equivalent to specifying STARTBDY=3).

,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.

Note:

1. Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW is still supported, but IBM recommends using LOC=24 instead.
2. LOC=24 should not be used to allocate disabled reference (DREF) storage. If issued in AMODE24, an abend B78 will result. In AMODE31, the LOC=24 parameter will be ignored, and the caller will be given an address above 16 megabytes.

LOC=(24,31)
Indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=(BELOW,ANY) is the same as specifying LOC=(24,31). LOC=(BELOW,ANY) is still supported, but IBM recommends using LOC=(24,31) instead.

LOC=(24,64)
Indicates that virtual storage is to be located below 16 megabytes and central storage can be located anywhere in 64-bit storage.

LOC=31
LOC=(31,31)
Indicate that virtual and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC =31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64)
Indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

Note: When you specify LOC=31, 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)
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
LOC=(EXPLICIT,64)
Indicate that virtual storage is to be located at the address specified on the INADDR parameter, and central storage can be located anywhere in 64-bit storage.

When you specify EXPLICIT on a request for storage from the same virtual page as previously requested storage, you must request it in the same key, subpool, and central storage area as on the previous storage request. For example, if you request virtual storage backed with central storage below 16 megabytes, any subsequent requests for storage from that virtual page must be specified as LOC=(EXPLICIT,24).

,INADDR=*stor addr*
Specifies the desired virtual address for the storage to be obtained. When you specify INADDR, you must specify EXPLICIT on the LOC parameter.

Note:

1. The address specified on INADDR must be on a doubleword boundary.
2. Make sure that the virtual storage address specified on INADDR and the central storage backing specified on the LOC=EXPLICIT parameter are a valid combination. For example, if the address specified on INADDR is for virtual storage above 16 megabytes, specify LOC=EXPLICIT or LOC=(EXPLICIT,ANY). Valid combinations include:
   - Virtual above, central any
   - Virtual any, central any
   - Virtual below, central below
   - Virtual below, central any

,CHECKZERO=YES
,CHECKZERO=NO
Specifies whether or not the return code for a successful completion should indicate if the system has cleared the requested storage to zeroes. When CHECKZERO=NO is specified or defaulted, the return code for a successful completion is 0. When CHECKZERO=YES is specified, the return code for a successful completion is X'14' if the system has cleared the requested storage to zeroes, and 0 if the system has not cleared the requested storage to zeroes.

There is no performance cost to specifying CHECKZERO=YES.

Programs that issue the GETMAIN macro with the CHECKZERO parameter can run on any z/OS system. On a down-level system, CHECKZERO will be ignored, and the return code for a successful completion (conditional or unconditional) will be 0.

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

ABEND codes

Abend codes the GETMAIN macro might issue are listed below in hexadecimal. For detailed abend code information, see *z/OS MVS System Codes*.

- 104
- 10A
- 178
- 204
- 20A
- 278
- 30A
- 378
- 40A
- 478
- 504
- 604
- 704
- 70A
- 778
- 804
- 80A
- 878
- 90A
- 978
- A0A
- A78
- B04
- B0A
- B78
- D04
- D0A
- D78

Return and reason codes

When the GETMAIN macro returns control to your program and you specified a conditional request, GPR 15 contains one of the following hexadecimal return codes:

Table 23. Return Codes for the GETMAIN Macro

Return Code	Meaning and Action
0	**Meaning**: Successful completion. CHECKZERO=YES was not specified, or the system has not cleared the requested storage to zeroes. **Action**: None.
4	**If you did not specify EXPLICIT on the LOC parameter**: • **Meaning**: Environmental or system error. Virtual storage was not obtained because insufficient storage is available. • **Action**: If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit. **If you specified EXPLICIT on the LOC parameter**: • **Meaning**: Program error. Virtual storage was not obtained because part of the requested storage area is outside the bounds of the user region. • **Action**: Determine why your program is mistakenly requesting storage outside the user region. If the request was for low private (local) storage, consult the system programmer to see if you have exceeded an installation-determined private storage limit.
8	**Meaning**: System error. Virtual storage was not obtained because the system has insufficient central storage to back the request. **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected.
C	**Meaning**: System error. Virtual storage was not obtained because the system cannot page in the page table associated with the storage to be allocated. **Action**: Report the problem to the system programmer so the cause of the problem can be determined and corrected.
10	**Meaning**: Program error. Virtual storage was not obtained for one of the following reasons: This reason code applies only to GETMAIN requests with LOC=EXPLICIT specified. • Part of the requested area is allocated already. • Virtual storage was already allocated in the same page as this request, but one of the following characteristics of the storage was different: – The subpool – The key – Central storage backing **Action**: Determine why your program is attempting to obtain allocated storage or why your program is attempting to obtain virtual storage with different attributes from the same page of storage. Correct the coding error.
14	**Meaning**: Successful completion. The system has cleared the requested storage to zeroes. This return code occurs only when CHECKZERO=YES is specified. **Action**: None.

Example 1

Obtain 400 bytes of storage from subpool 10. If the storage is available, the address will be returned in register 1 and register 15 will contain 0; if storage is not available, register 15 will contain 4.

```
GETMAIN   RC,LV=400,SP=10
```

Example 2

Obtain 48 bytes of storage from default subpool 0. If the storage is available, the address will be stored in the word at AREAADDR; if the storage is not available, the task will be abnormally terminated.

```
         GETMAIN EU,LV=48,A=AREAADDR
         .
         .
         .
AREAADDR DS      F
```

Example 3

Obtain a minimum of 1024 bytes to a maximum of 4096 bytes of virtual storage from default subpool 0 with virtual and central storage locations either above or below 16 megabytes. If the storage is available, the starting address is to be returned in register 1 and the length of the storage allocated is to be returned in register 0; if the storage is not available, the caller is to be terminated.

```
GETMAIN VRU,LV=(4096,1024),LOC=ANY
```

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:

Syntax	Description
name	*name*: Begin *name* in column 1.
ƀ	One or more blanks must precede GETMAIN.
GETMAIN	
ƀ	One or more blanks must follow GETMAIN.
LC	
LU	
VC	
VU	
EC	
EU	

Syntax	Description
,LA=*length addr*	*length addr*: A-type address.
,LV=*length value*	*length value*: symbol or decimal number. **Note:** 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	**Default**: BNDRY=DBLWD
,BNDRY=PAGE	
,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:

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

Syntax	Description
ƀ	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). **Note:** 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	
,RELATED=*value*	*value*: any valid assembler character string.
,MF=(E,*list addr*)	*ctrl prog*: 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:

,MF=(E,*list addr*)
Specifies the execute form of the GETMAIN macro using a remote control program parameter list.

Chapter 101. GTZQUERY macro — GTZ Query

Description

The GTZQUERY macro provides the interface to obtain various pieces of information about the status of the Generic Tracker for z/OS, its configuration, and any stored data.

Additional references and an overview for the tracking facility can be found in *z/OS MVS Diagnosis: Tools and Service Aids*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. PSW key 8-15 When problem state, key 8-15 and not APF authorized, the caller needs to be authorized for READ access to the XFACILIT class resource GTZ.sysname.QUERY.
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro. Use of the special ALET value of 1 ("secondary") for addressing parameters is not recommended and might be rejected via a "Bad ALET" reason code since a space switch might lead to loss of addressability of such a parameter.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). Control parameters above 2GB are allowed only for AMODE 64 callers.

Programming Requirements

For Assembler, include the GTZZQRY macro to get a mapping for the answer area and to get equate symbols for related constants as well as return and reason codes.

For (METAL-) C use include file gtzhqry.h instead of GTZZQRY.

Restrictions

The caller must not have a functional recovery routine (FRR) established.

Input Register Information

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

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

Output Register Information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, when register 15 is not 0.

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance Implications

None.

Syntax

The GTZQUERY macro is written as follows:

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

Syntax	Description
GTZQUERY	
ƀ	One or more blanks must follow GTZQUERY.
REQUEST=STATUS	
REQUEST=EXCLUDE	
REQUEST=DEBUG	
REQUEST=TRACKDATA	
,OWNER=*owner*	*owner:* RS-type address or address in register (2) - (12)
,OWNER=*	**Default:** OWNER=*
,SOURCETYPE=ALL	**Default:** SOURCETYPE=ALL
,SOURCETYPE=NOPATH	
,SOURCETYPE=PATH	
,SOURCE=*source*	*source*: RS-type address or address in register (2) - (12)
,SOURCE=*	**Default:** SOURCE=*
,SOURCEPATHLEN=*sourcepathlen*	*sourcepathlen:* RS-type address or address in register (2) - (12)
,SOURCEPATHLEN=0,	**Default:** SOURCEPATHLEN=0,
,SOURCEPATH=*sourcepath*	*sourcepath:* RS-type address or address in register (2) - (12)
,EVENTDESCLEN=*eventdesclen*	*eventdesclen:* RS-type address or address in register (2) - (12)
,EVENTDESCLEN=0	**Default:** EVENTDESCLEN=0
,EVENTDESC=*eventdesc*	*eventdesc:* RS-type address or address in register (2) - (12)
,EVENTDATA=*eventdata*	*eventdata:* RS-type address or address in register (2) - (12)
,EVENTDATA=ALL	**Default:** EVENTDATA=ALL
,EVENTASID=*eventasid*	*eventasid:* RS-type address or address in register (2) - (12)
,EVENTASID=ALL	**Default:** EVENTASID=ALL
,EVENTJOB=*eventjob*	*eventjob:* RS-type address or address in register (2) - (12)
,EVENTJOB=*	**Default:** EVENTJOB=*
,PROGRAMTYPE=ALL	**Default:** PROGRAMTYPE=ALL

Syntax	Description
,PROGRAMTYPE=NOPATH	
,PROGRAMTYPE=PATH	
,PROGRAM=*program*	*program:* RS-type address or address in register (2) - (12)
,PROGRAM=*	**Default:** PROGRAM=*
,PROGRAMPATHLEN=*programpathlen*	*programpathlen* RS-type address or address in register (2) - (12)
,PROGRAMPATHLEN=0	**Default:** PROGRAMPATHLEN=0,
,PROGRAMPATH=*programpath*	*programpath:* RS-type address or address in register (2) - (12)
,PROGRAMOFFSET=*programoffset*	*programoffset:* RS-type address
,PROGRAMOFFSET=ALL	**Default:** PROGRAMOFFSET=ALL
,HOMEASID=*homeasid*	*homeasid:* RS-type address or address in register (2) - (12)
,HOMEASID=ALL	**Default:** HOMEASID=ALL
,HOMEJOB=*homejob*	*homejob:* RS-type address or address in register (2) - (12)
,HOMEJOB=*	**Default:** HOMEJOB=*
,ANSAREA=*ansarea*	*ansarea:* RS-type address or address in register (2) - (12)
,ANSLEN=*anslen*	*anslen:* RS-type address or address in register (2) - (12)
,SECCHECK=UNAUTH	**Default:** SECCHECK=UNAUTH
,SECCHECK=ALL	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15), (REG15), or (R15).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (0) or (2) - (12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S

Syntax	Description
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*, COMPLETE))	
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained as follows:

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

REQUEST=STATUS
REQUEST=EXCLUDE
REQUEST=DEBUG
REQUEST=TRACKDATA
A required parameter, which identifies the type of request.

REQUEST=STATUS
Get general status information for the tracking facility and an overview of current configuration settings and accumulated tracking data.

REQUEST=EXCLUDE
Retrieve detail information about currently active exclusion filters. Compare for example the SETGTZ EXCLUDE command.

REQUEST=DEBUG
Retrieve detail information about currently active debug action filters. Compare for example the SETGTZ DEBUG command.

REQUEST=TRACKDATA
Retrieve individual unique tracked instances, as created by GTZTRACK calls, and their associated statistics as currently stored by the tracking facility.

,OWNER=*owner*
,OWNER=***
When REQUEST=TRACKDATA is specified, an optional input parameter, which requests to return only such tracked instances which have a matching OWNER value, as specified on the GTZTRACK invocations that recorded the tracked instances.

If OWNER is specified, the owner value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- will be compared case-insensitive and will have trailing blanks ignored in comparisons.
- can only contain alphabetic characters (A-Z,a-z), numerics (0-9), national characters (@,$,#), the underscore ('_'), a period (.), a dash (-), a slash (/), wildcards, or trailing blanks.

The default is * which indicates, as a single wildcard asterisk, to match any tracked instance's OWNER value.

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

,SOURCETYPE=ALL
,SOURCETYPE=NOPATH
,SOURCETYPE=PATH

When REQUEST=TRACKDATA is specified, an optional parameter, which identifies what type of tracked instance source values to match. The default is SOURCETYPE=ALL.

,SOURCETYPE=ALL
Match any source value, regardless if the tracked instance has a SOURCE or SOURCEPATH asociated with it.

,SOURCETYPE=NOPATH
Match only tracked instances which have a matching SOURCE value. Tracked instances with a SOURCEPATH value will not be matched.

,SOURCETYPE=PATH
Match only tracked instances which have a matching SOURCEPATH value. Tracked instances with a SOURCE value will not be matched.

,SOURCE=*source*
,SOURCE=***

When SOURCETYPE=NOPATH and REQUEST=TRACKDATA are specified, an optional input parameter, which requests to return only such tracked instances which have a matching SOURCE value.

The source value here:

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- can only contain alphabetic characters (A-Z,a-z), numerics (0-9), national characters (@,$,#), wildcards, or trailing blanks.
- will be converted to upper-case for comparisons and will have trailing blanks ignored in comparisons.

The default is * , a single wildcard asterisk, which indicates to match all tracked instances with a SOURCE value.

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

,SOURCEPATHLEN=*sourcepathlen*
,SOURCEPATHLEN=0

When SOURCEPATH=PATH and REQUEST=TRACKDATA are specified, an optional input parameter, which specifies the length of the SOURCEPATH to match. SOURCEPATHLEN must be in the range of 0 through 1024.

The default is 0, which indicates to match any SOURCEPATH value.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *sourcepathlen* must be in the range 0 through 1024.

,SOURCEPATH=*sourcepath*

When SOURCEPATHLEN=*sourcepathlen*, SOURCETYPE=PATH and REQUEST=TRACKDATA are specified, a required input parameter, which requests to return only such tracked instances which have a matching SOURCEPATH value with a length as specified by SOURCEPATHLEN.

The source path value here:

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- can contain any EBCDIC character, but not just blanks or just binary zeroes. IBM recommends you only use printable characters.
- will not have its alphabetic characters folded to upper-case and will be compared case-sensitive.

- will not have its trailing blanks removed for comparisons.

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

,EVENTDESCLEN=*eventdesclen*
,EVENTDESCLEN=0
When REQUEST=TRACKDATA is specified, an optional input parameter, which specifies the length of the event description (EVENTDESC) to match. EVENTDESCLEN must be in the range 0 through 64.

The default is 0 which indicates to match any EVENTDESC value.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *eventdesclen* must be in the range 0 through 64.

,EVENTDESC=*eventdesc*
When EVENTDESCLEN=*eventdesclen* and REQUEST=TRACKDATA are specified, a required input parameter, which requests to return only such tracked instances which have a matching event description (EVENTDESC) value with a length as specified by EVENTDESCLEN.

The EVENTDESC value:

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- can contain any EBCDIC character, but not just blanks or just binary zeroes. IBM recommends to only use printable characters.
- will not have its alphabetic characters folded to upper-case and will be compared case-sensitive.
- will not have its trailing blanks removed.

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

,EVENTDATA=*eventdata*
,EVENTDATA=ALL
When REQUEST=TRACKDATA is specified, an optional input parameter, which requests to return only such tracking instances which have a matching EVENTDATA value as specified on the GTZTRACK invocations that recorded the tracked instances.

The default is ALL which indicates to match all EVENTDATA values.

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

,EVENTASID=*eventasid*
,EVENTASID=ALL
When REQUEST=TRACKDATA is specified, an optional input parameter, which requests to return only such tracked instances which have a matching EVENTASID value as specified on the GTZTRACK invocations that recorded the tracked instances. (address space identifier)

The default is ALL which indicates to match all EVENTASID values.

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

,EVENTJOB=*eventjob*
,EVENTJOB=***
When REQUEST=TRACKDATA is specified, an optional input parameter, which requests to return only such tracking instances which were associated with a matching job name of the address space identified by the EVENTASID value specified on the GTZTRACK invocations that recorded the tracked instances.

If EVENTJOB is specified, the *eventjob* value

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- can only contain alphabetic characters (A-Z,a-z), numerics (0-9), national characters (@,$,#), wildcards, or trailing blanks.

- will be converted to upper-case for comparisons and will have trailing blanks ignored in comparisons.

The default is * which indicates, as a single wildcard asterisk, to match. any tracked instance's EVENTJOB value.

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

PROGRAMTYPE=ALL
PROGRAMTYPE=NOPATH
PROGRAMTYPE=PATH

When REQUEST=TRACKDATA is specified, an optional parameter, which identifies what type of tracked instance program values, as derived from the EVENTADDR value specified on the GTZTRACK invocations that recorded the tracked instance, to match. A tracked instance is associated with a PROGRAMPATH value if the derived program is a z/OS Unix program, otherwise the tracked instance is associated with a PROGRAM value. The default is PROGRAMTYPE=ALL.

PROGRAMTYPE=ALL
Match any program value, regardless if the tracked instance has a PROGRAM or PROGRAMPATH associated with it.

,PROGRAMTYPE=NOPATH
Match only tracked instances which have a matching PROGRAM value. Tracked instances with a PROGRAMPATH value will not be matched.

,PROGRAMTYPE=PATH
Match only tracked instances which have a matching PROGRAMPATH value. Tracked instances with a PROGRAM value will not be matched.

,PROGRAM=*program*
,PROGRAM=***

When PROGRAMTYPE=NOPATH and REQUEST=TRACKDATA are specified, an optional input parameter, which requests to return only such tracked instances which have a matching PROGRAM value.

The program value here:

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- can only contain alphabetic characters (A-Z,a-z), numerics (0-9), national characters (@,$,#), wildcards, or trailing blanks.
- will be converted to upper-case for comparisons and will have trailing blanks ignored in comparisons.

The default is * , a single wildcard asterisk, which indicates to match all tracked instances with a PROGRAM value.

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

,PROGRAMPATHLEN=*programpathlen*
,PROGRAMPATHLEN=0

When PROGRAMTYPE=PATH and REQUEST=TRACKDATA are specified, an optional input parameter, which specifies the length of the PROGRAMPATH to match. PROGRAMPATHLEN must be in the range 0 through 1024.

The default is 0, which indicates to match any PROGRAMPATH value.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *programpathlen* must be in the range of 0 through 1024.

,PROGRAMPATH=*programpath*

When PROGRAMPATHLEN=programpathlen, PROGRAMTYPE=PATH and REQUEST=TRACKDATA are specified, a required input parameter, which requests to return only such tracked instances which have a matching PROGRAMPATH value with a length as specified by PROGRAMPATHLEN.

The program path value here:

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- can contain any EBCDIC character, but not just blanks or just binary zeroes. IBM recommends to only use printable characters.
- will not have its alphabetic characters folded to upper-case and will be compared case-sensitive.
- will not have its trailing blanks removed for comparisons.

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

,PROGRAMOFFSET=*programoffset*
,PROGRAMOFFSET=ALL

When REQUEST=TRACKDATA is specified, an optional input parameter, which requests to return only such tracked instances which were associated with a matching program offset as derived from the EVENTADDR value specified on the GTZTRACK invocations that recorded the tracked instances.

The default is ALL which indicates to match all PROGRAMOFFSET values.

To code: Specify the RS-type address of a doubleword field, or specify a literal decimal value.

,HOMEASID=*homeasid*
,HOMEASID=ALL

When REQUEST=TRACKDATA is specified, an optional input parameter, which requests to return only such tracked instances which were associated with a matching ASID of the HOME address space at the time of the GTZTRACK invocations that recorded the tracked instances.

The default is ALL which indicates to match all HOME ASID values.

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

,HOMEJOB=*homejob*
,HOMEJOB=***

When REQUEST=TRACKDATA is specified, an optional input parameter, which requests to return only such tracked instances which were associated with a matching job name of the HOME address space at the time of the GTZTRACK invocations that recorded the tracked instances.

If HOMEJOB is specified, the homejob value:

- can contain wildcards anywhere in the string: An asterisk (*) will match any character sequence of zero or more characters and a question mark (?) will match a single arbitrary character.
- can only contain alphabetic characters (A-Z,a-z), numerics (0-9), national characters (@,$,#), wildcards, or trailing blanks.
- will be converted to upper-case for comparisons and will have trailing blanks ignored in comparisons.

The default is * which indicates, as a single wildcard asterisk, to match. any tracked instance's HOMEJOB value.

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

,ANSAREA=*ansarea*

A required output parameter. A parameter which is to contain the returned information. The length of ANSAREA is given via ANSLEN. The answer area is mapped by macro GTZZQRY and has to start at a double-word boundary.

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

,ANSLEN=*anslen*

A required input parameter, which contains the length of the provided answer area in bytes. The length must be at least of value GtzQuaaMinAnsLen, which is defined in macro GTZZQRY.

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

,SECCHECK=UNAUTH
,SECCHECK=ALL
An optional parameter that indicates whether to do RACF security checks based on profile(s) GTZ.sysname.QUERY. If RACF does not grant authority, the request is rejected. The default is SECCHECK=UNAUTH.

,SECCHECK=UNAUTH
that indicates to do RACF security checks only when the caller is unauthorized (not supervisor state, not system key, not APF-authorized).

,SECCHECK=ALL
that indicates to do RACF security checks in all cases.

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)

An optional input parameter that specifies the macro form.

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

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

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

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

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

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

,list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

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

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

ABEND Codes

E77

A GTZ ABEND might be issued by the system in error situations.

In the following GtzQuery abend reason codes, the two bytes designated by "xxxx" are for the tracker component's diagnostic purposes and have no significance to the external interface.

Reason Code

(Hex)
Explanation

xxxxyyyy
Report this to the system programmer to contact IBM Service.

Return and Reason Codes

When the GTZQUERY macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro GTZZQRY provides equate symbols for the return and reason codes.

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

Table 24. Return and Reason Codes for the GTZQUERY Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	–	**Equate Symbol**: GtzQueryRc_OK **Meaning**: Successfully returned requested information. **Action**: None required
8	–	**Equate Symbol**: GtzQueryRc_Error **Meaning**: Error **Action**: Refer to action under the individual reason code.
8	xxxx0880	**Equate Symbol**: GtzQueryRsn_BadRequest **Meaning**: A bad REQUEST type has been specified. **Action**: Use one of the supported request types.
8	xxxx0881	**Equate Symbol**: GtzQueryRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure the ALET associated with the parameter list is valid. The access register might not have been set up correctly.
8	xxxx0882	**Equate Symbol**: GtzQueryRsn_BadParmlist **Meaning**: Error accessing the parameter list. **Action**: Make sure the provided parameter list is valid.
8	xxxx0883	**Equate Symbol**: GtzQueryRsn_BadParmlistVersion **Meaning**: The specified version of the macro is not compatible with the current version of IBM Generic Tracker for z/OS. **Action**: Avoid requesting parameters that are not supported by this version of IBM Generic Tracker for z/OS.

Table 24. Return and Reason Codes for the GTZQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0884	**Equate Symbol**: GtzQueryRsn_BadAnsAreaALET **Meaning**: Bad ANSAREA ALET. **Action**: Make sure the ALET associated with the answer area is valid. The access register might not have been set up correctly.
8	xxxx0885	**Equate Symbol**: GtzQueryRsn_BadAnsAreaAddrNull **Meaning**: ANSAREA address is NULL. **Action**: Check the location of your answer area. Typically address zero is not a valid address.
8	xxxx0886	**Equate Symbol**: GtzQueryRsn_BadAnsAreaAddrAlign **Meaning**: The ANSAREA has a bad alignment. **Action**: The ANSAREA has to start at a double-word boundary.
8	xxxx0887	**Equate Symbol**: GtzQueryRsn_BadAnsLen **Meaning**: Bad ANSLEN value. **Action**: Provide an answer area which is at least GtzQuaaMinAnsLen bytes long.
8	xxxx0888	**Equate Symbol**: GtzQueryRsn_BadAnsArea **Meaning**: Error accessing answer area. **Action**: Make sure the provided answer area is valid.
8	xxxx0889	**Equate Symbol**: GtzQueryRsn_BadSecCheckValue **Meaning**: Bad SECCHECK value. **Action**: Specify a support SECCHECK value.
8	xxxx088A	**Equate Symbol**: GtzQueryRsn_BadEnvNotEnabled **Meaning**: Not enabled. **Action**: Avoid using GTZQUERY when not enabled for I/O and external interrupts
8	xxxx088B	**Equate Symbol**: GtzQueryRsn_BadEnvLocked **Meaning**: Locked. **Action**: Avoid using GTZQUERY when a lock is held.
8	xxxx088C	**Equate Symbol**: GtzQueryRsn_BadEnvSrbmode **Meaning**: SRB mode. **Action**: Avoid issuing GTZQUERY in SRB mode.
8	xxxx088D	**Equate Symbol**: GtzQueryRsn_BadEnvFRR **Meaning**: The caller had an EUT FRR established. **Action**: Avoid using HZSPWRIT when an EUT FRR is established.

Table 24. Return and Reason Codes for the GTZQUERY Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
8	xxxx088E	**Equate Symbol**: GtzQueryRsn_BadEnvNotInGtz **Meaning**: The processing module for GTZQUERY has been invoked outside of the GTZ address space. **Action**: Use the provided GTZQUERY macro to call the processing module.
8	xxxx088F	**Equate Symbol**: GtzQueryRsn_NotAuthorized **Meaning**: Not authorized. **Action**: Ensure you are authorized to perform the requested operation.
8	xxxx0890	**Equate Symbol**: GtzQueryRsn_FacilityNotAvailable **Meaning**: Generic Tracker is not available. **Action**: This might be a temporary situation. If this problem persists, contact IBM Support.
8	xxxx0891	**Equate Symbol**: GtzQueryRsn_BadOwnerCharset **Meaning**: The OWNER parameter value contains bad characters. **Action**: Use only allowed characters as documented for the OWNER parameter.
8	xxxx0892	**Equate Symbol**: GtzQueryRsn_BadSourcePathALET **Meaning**: Bad SOURCEPATH ALET. **Action**: Make sure that the ALET associated with the SOURCEPATH parameter is valid. The access register might not have been set up correctly.
8	xxxx0893	**Equate Symbol**: GtzQueryRsn_BadSourcePath **Meaning**: Error accessing SOURCEPATH. **Action**: Make sure that the provided SOURCEPATH is properly addressable.
8	xxxx0894	**Equate Symbol**: GtzQueryRsn_BadProgramPathALET **Meaning**: Bad PROGRAMPATH ALET. **Action**: Make sure that the ALET associated with the PROGRAMPATH parameter is valid. The access register might not have been set up correctly.
8	xxxx0895	**Equate Symbol**: GtzQueryRsn_BadProgramPath **Meaning**: Error accessing PROGRAMPATH. **Action**: Make sure that the provided PROGRAMPATH is properly addressable.

Table 24. Return and Reason Codes for the GTZQUERY Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0896	**Equate Symbol**: GtzQueryRsn_BadEventDescALET **Meaning**: Bad EVENTDESC ALET. **Action**: Make sure that the ALET associated with the EVENTDESC parameter is valid. The access register might not have been set up correctly.
8	xxxx0897	**Equate Symbol**: GtzQueryRsn_BadEventDesc **Meaning**: Error accessing EVENTDESC. **Action**: Make sure that the provided EVENTDESC is properly addressable.
8	xxxx0898	**Equate Symbol**: GtzQueryRsn_BadProgramCharset **Meaning**: The PROGRAM parameter value contains bad characters. **Action**: Use only allowed characters as documented for the PROGRAM parameter.
8	xxxx0899	**Equate Symbol**: GtzQueryRsn_BadProgramPathCharset **Meaning**: The PROGRAMPATH parameter value contains bad characters. **Action**: Use only allowed characters as documented for the PROGRAMPATH parameter.
8	xxxx089A	**Equate Symbol**: GtzQueryRsn_BadSourceCharset **Meaning**: The SOURCE parameter value contains bad characters. **Action**: Use only allowed characters as documented for the SOURCE parameter.
8	xxxx089B	**Equate Symbol**: GtzQueryRsn_BadSourcePathCharset **Meaning**: The SOURCEPATH parameter value contains bad characters. **Action**: Use only allowed characters as documented for the SOURCEPATH parameter.
8	xxxx089C	**Equate Symbol**: GtzQueryRsn_BadEventDescCharset **Meaning**: The EVENTDESC parameter value contains bad characters. **Action**: Use only allowed characters as documented for the EVENTDESC parameter.
8	xxxx089D	**Equate Symbol**: GtzQueryRsn_BadEventDescLen **Meaning**: The EVENTDESCLEN parameter value is out of range. **Action**: Specify an EVENTDESCLEN in the documented allowed range.

Table 24. Return and Reason Codes for the GTZQUERY Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
8	xxxx089E	**Equate Symbol**: GtzQueryRsn_BadSourcePathLen **Meaning**: The SOURCEPATHLEN parameter value is out of range. **Action**: Specify an SOURCEPATHLEN in the documented allowed range.
8	xxxx089F	**Equate Symbol**: GtzQueryRsn_BadProgramPathLen **Meaning**: The PROGRAMPATHLEN parameter value is out of range. **Action**: Specify an PROGRAMPATHLEN in the documented allowed range.
8	xxxx08A0	**Equate Symbol**: GtzQueryRsn_BadProgramType **Meaning**: Invalid PROGRAMTYPE. **Action**: Use only documented PROGRAMTYPE values.
8	xxxx08A1	**Equate Symbol**: GtzQueryRsn_BadSourceType **Meaning**: Invalid SOURCETYPE. **Action**: Use only documented SOURCETYPE values.
8	xxxx08A2	**Equate Symbol**: GtzQueryRsn_BadHomeJobCharset **Meaning**: The HOMEJOB parameter value contains bad characters. **Action**: Use only allowed characters as documented for the HOMEJOB parameter.
8	xxxx08A3	**Equate Symbol**: GtzQueryRsn_BadEventJobCharset **Meaning**: The EVENTJOB parameter value contains bad characters. **Action**: Use only allowed characters as documented for the EVENTJOB parameter.
0C	–	**Equate Symbol**: GtzQueryRc_SevereError **Meaning**: Severe Error / Environment Error **Action**: Refer to action under the individual reason code.
0C	xxxx0C90	**Equate Symbol**: GtzQueryRsn_FacilityNotAvailable **Meaning**: Generic Tracker is not available. **Action**: This might be a temporary situation. See the description of message GTZ1000I for further information.
0D	-	**Equate Symbol**: GtzQueryRc_OutOfMemory **Meaning**: Tracking facility is low on memory. **Action**: See the description of message GTZ0004E. Try also to omit any filters, for example for REQUEST(TRACKDATA).

Table 24. Return and Reason Codes for the GTZQUERY Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
10	–	**Equate Symbol**: GtzQueryRc_CompError **Meaning**: Component error. **Action**: Report the associated reason code to the system programmer to contact IBM Service.

Examples

See the sample health check GTZSHCK in SYS1.SAMPLIB for sample GTZQUERY invocations.

Chapter 102. GTZTRACK macro — GTZ Track

Description

The GTZTRACK macro provides the interface to request caller information, an "event", to be recorded by the Generic Tracker for z/OS, as a "tracked instance".

Additional references and an overview for the tracking facility can be found in *z/OS MVS Diagnosis: Tools and Service Aids*.

For GTZTRACK the caller does not have to determine first whether the tracking facility is present or if tracking is enabled or if the tracking facility is full. The service will handle those situations.

The following macro parameters, specified explicitly or implicitly derived at GTZTRACK call time, will be used to uniquely identify a tracked instance. If two GTZTRACK request have the same unique key parameters, only one tracked instance will be recorded and only the associated occurrence count will be incremented for the second request. These unique parameters can also be used later to include only certain subsets of tracked instances for certain actions, like for excluding such instances from being recorded at all, or for reporting. The unique parameters are:

- OWNER
- SOURCE / SOURCEPATH
- EVENTDESC
- the event program (/-path) name and program offset, as derived from the EVENTADDR
- the home address space job name and associated ASID, as derived from the dispatchable unit (task...) in which GTZTRACK was invoked
- the event address space job name and associated ASID, as derived from the EVENTASID
- the authorization state, as derived from the EVENTPSW8/16. "Not Authorized" when running in problem state, key 8-15, and not APF authorized, otherwise "Authorized".
- EVENTDATA

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state. PSW key 8-15 Callers in problem state, key 8-15, and not APF authorized: • are limited to a maximum of eight unique tracked instances being recorded per associated HOME address space of the caller. Any further track requests beyond that limit will be ignored • are limited to use EVENTASID(HOME) and are not allowed to use any other ASID values
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit If in AMODE 64, specify SYSSTATE AMODE64=YES before invoking this macro.

Environmental factor	Requirement
ASC mode:	Primary or access register (AR) If in Access Register ASC mode, specify SYSSTATE ASCENV=AR before invoking this macro. Use of the special ALET value of 1 ("secondary") for addressing parameters is not recommended and might be rejected via a "Bad ALET" reason code since a space switch might lead to loss of addressability of such a parameter.
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). Control parameters above 2GB are allowed only for AMODE 64 callers.

Programming Requirements

The caller can include the GTZZTRK macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have an FRR established.

Input Register Information

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

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

Output Register Information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, when register 15 is not 0.

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as work registers by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance Implications

The GTZTRACK service is designed to keep its synchronously executed code path short. For that purpose part of the request processing is queued to be executed asynchronously in the GTZ address space.

The macro expansion will also ensure a minimal code path and skip the call to the synchronous backend processing, if tracking is not enabled or if the facility is full or too busy for additional requests.

Syntax

The GTZTRACK macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GTZTRACK.
GTZTRACK	
ƀ	One or more blanks must follow GTZTRACK.
OWNER=*owner*	*owner:* RS-type address or address in register (2) - (12)
,SOURCE=*source*	*source:* RS-type address or address in register (2) - (12)
,SOURCEPATH=*sourcepath*	*sourcepath:* RS-type address or address in register (2) - (12)
,SOURCEPATHLEN=*sourcepathlen*	*sourcepathlen:* RS-type address or address in register (2) - (12)
,EVENTDESC=*eventdesc*	*eventdesc:* RS-type address or address in register (2) - (12)
,EVENTDESCLEN=*eventdesclen*	*eventdesclen:* RS-type address or address in register (2) - (12)
,EVENTDATA=*eventdata*	*eventdata:* RS-type address or address in register (2) - (12)
,EVENTDATA=ALLZER0	**Default:** EVENTDATA=ALLZERO

Syntax	Description
,EVENTADDR=*eventaddr*	*eventaddr:* RS-type address or address in register (2) - (12)
,EVENTPSW16=*eventpsw16*	*eventpsw16:* RS-type address or address in register (2) - (12)
,EVENTPSW8=*eventpsw8*	*eventpsw8:* RS-type address or address in register (2) - (12)
,EVENTASID=HOME	
,EVENTASID=PRIMARY	
,EVENTASID=SECONDARY	
,EVENTASID=VALUE	
,EVENTASIDVAL=*eventasidval*	*eventasidval:* RS-type address
,NOABEND=NO	**Default:** NOABEND=NO
,NOABEND=YES	
,NOABEND=VALUE	
,NOABENDVAL=*noabendval*	*noabendval:* RS-type address
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12) or (15), (GPR15), (REG15), or (R15).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (0) or (2) - (12), (00), (GPR0), (GPR00), (REG0), (REG00), or (R0).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Syntax	Description
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*	
,MF=(M,*list addr*, COMPLETE))	
,MF=(M,*list addr*,NOCHECK)	

Parameters

The parameters are explained as follows:

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

OWNER=*owner*
A required input parameter, which specifies the owner of the code issuing GTZTRACK. IBM recommends that you use your company name, followed by a short component name, for example IBMGRS, as the owner.

Only alphabetic characters (A-Z,a-z), numerics (0-9), national characters (@,$,#), an underscore ('_'), a period (.), a dash (-), a slash (/), and trailing blanks are allowed.

The owner value will be treated case-insensitive in future filter comparisons, but will have its case preserved in any output reports.

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

,SOURCE=*source*
,SOURCEPATH=*sourcepath*
A required input parameter.

,SOURCE=*source*
A parameter which further identifies the code which is issuing GTZTRACK, as a sub-qualification of the OWNER in form of for example the name of the "(detecting) module" which issued GTZTRACK.

Only alphabetic characters (A-Z,a-z), numerics (0-9), national characters (@,$,#), and trailing blanks are allowed.

The system will convert this value to upper-case and use it this way in future filter comparisons and any output reports.

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

,SOURCEPATH=*sourcepath*
A parameter which further identifies a z/OS Unix caller of this service, as a sub-qualification of the OWNER. The length of the SOURCEPATH is given via SOURCEPATHLEN.

Any character is allowed in the source path. It can not be all blanks or all binary zeroes though. IBM recommends to only use printable characters.

Character case and trailing blanks are both preserved and are significant in comparisons.

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

,SOURCEPATHLEN=*sourcepathlen*
When SOURCEPATH=*sourcepath* is specified, a required input parameter, which specifies the length of the SOURCEPATH. SOURCEPATHLEN must be in the range 1 through 1024.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *sourcepathlen* must be in the range 1 through 1024.

,EVENTDESC=*eventdesc*
A required input parameter, which describes the tracked instance.

Any character is allowed in the event description, but it can not contain just blanks or just binary zeroes. IBM recommends to only use printable characters.

Character case and trailing blanks are both preserved and are significant in comparisons.

The length of the EVENTDESC field is given via EVENTDESCLEN.

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

,EVENTDESCLEN=*eventdesclen*
A required input parameter, which specifies the length of the EVENTDESC field. EVENTDESCLEN must be in the range 1 through 64.

To code: Specify the RS-type address, or address in register (2)-(12), of a halfword field, or specify a literal decimal value. *eventdesclen* must be in the range 1 through 64.

,EVENTDATA=*eventdata*
,EVENTDATA=ALLZERO
An optional input parameter, which specifies data associated with this tracked instance. Callers can use it for their own purposes, for example to pass dynamic parameters of a service being tracked. Any display output will format this data as a 16-byte hexadecimal number.

The default is ALLZERO.

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

,EVENTADDR=*eventaddr*
,EVENTPSW16=*eventpsw16*
A required input parameter.

,EVENTADDR=*eventaddr*
A parameter that contains the address of where the event being tracked occurred. A typical value would be the address to which the code invoking GTZTRACK will return.

This address is assumed to be a 64-bit address. If a 24-bit or 31-bit address is provided, you must ensure that the appropriate high-order bits and bytes of the address are set to zero. For example you should not supply an 8-byte address that is a "pointer-defined" AMODE 31 address, such as 00000000_81234568, which in other circumstances might indicate address 1234568 with the AMODE 31 bit on.

The GTZTRACK service will attempt to translate this address to the name of the program that is loaded at that address in memory. The program name will be stored by the GTZTRACK service for later retrieval. For example, the DISPLAY GTZ,TRACKDATA command shows this information via the PROGRAM, or PROGRAMPATH, output field of message GTZ1002I for each displayed track event.

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

,EVENTPSW16=*eventpsw16*
A parameter character input that contains a z/Architecture 16-byte Program Status Word (PSW). The PSW contains an instruction address which is being used as the address of where the event being tracked occurred. The GTZTRACK service will attempt to translate this address to the name of the program that is loaded at that address in memory. A typical value would be the address to which the code invoking GTZTRACK will return. Besides the instruction address, GTZTRACK will also extract from the given PSW the bits that describe the program key and the problem state status. This information, along with the current APF authorization of the GTZTRACK caller, will be used to categorize the track event as "authorized" or "not authorized". Both the program name and the authorization status will be stored by the GTZTRACK service for later retrieval. For example, the DISPLAY GTZ,TRACKDATA command shows this information via the PROGRAM, or PROGRAMPATH, and the AUTHORIZED output fields of message GTZ1002I for each displayed track event.

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

,EVENTPSW8=*eventpsw8*

When EVENTADDR=eventaddr is specified, a required 8 character input that contains the 8-byte Program Status Word (PSW) to be associated with this track event. GTZTRACK will extract from this PSW the bits that describe the program key and the problem state status. This information, along with the current APF authorization of the GTZTRACK caller, will be used to categorize the track event as "authorized" or "not authorized". This authorization status will be stored by the GTZTRACK service for later retrieval. For example, the DISPLAYGTZ,TRACKDATA command shows this information via the AUTHORIZED output field of message GTZ1002I for each displayed track event.

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

,EVENTASID=HOME
,EVENTASID=PRIMARY
,EVENTASID=SECONDARY
,EVENTASID=VALUE

A required parameter that specifies which address space to associate with the tracked instance. This is mainly useful for cross-memory cases, where the event occurred in one address space, but the GTZTRACK call in another and the event ASID is different than HOME.

,EVENTASID=HOME
the event occurred in the home address space

,EVENTASID=PRIMARY
the event occurred in the primary address space. Only authorized callers of GTZTRACK are allowed to specify this option.

,EVENTASID=SECONDARY
the event occurred in the secondary address space. Only authorized callers of GTZTRACK are allowed to specify this option.

,EVENTASID=VALUE
the event occurred in the address space identified via EVENTASIDVAL. Only authorized callers of GTZTRACK are allowed to specify this option.

,EVENTASIDVAL=*eventasidval*

When EVENTASID=VALUE is specified, a required input parameter that specifies the address space id to associate with the tracked instance.

To code: Specify the RS-type address of a 16 bit field.

,NOABEND=NO
,NOABEND=YES
,NOABEND=VALUE

An optional parameter that indicates whether to disallow diagnostic ABENDs for this track request. Some GTZTRACK callers might not be tolerant to ABENDs at all and can specify this as an override for any GTZPRMxx statement which might be matching this GTZTRACK call and which requests an ABEND. The default is NOABEND=NO.

,NOABEND=NO
that indicates diagnostic ABEND E77 is allowed for this track request, if a matching GTZPRMxx statement is found.

,NOABEND=YES
that indicates the tracking facility will never issue ABEND E77 for this track request for diagnostic purposes, even if requested by a matching GTZPRMxx statement.

,NOABEND=VALUE
use the NoAbendVal value.

,NOABENDVAL=*noabendval*

When NOABEND=VALUE is specified, a required input parameter that indicates whether to disallow diagnostic ABENDs or not. A value of one (1) means 'no abend allowed' and a zero (0) value means 'abends allowed'.

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

,RETCODE=*retcode*
An optional output parameter into which the return code is to be copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value will be left in GPR 15.

To code: Specify the RS-type address of a fullword field, or register (2)-(12) or (15), (GPR15), (REG15), or (R15).

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value will be left in GPR 0.

To code: Specify the RS-type address of a fullword field, or register (0) or (2)-(12), (00), (GPR0), (GPR00), REG0), (REG00), or (R0).

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

- **IMPLIED_VERSION**, which is the lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.
- **MAX**, if you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,MF=(E,*list addr*,NOCHECK)
,MF=(M,*list addr*)
,MF=(M,*list addr*,COMPLETE)
,MF=(M,*list addr*,NOCHECK)
An optional input parameter that specifies the macro form.

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

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

Use MF=E to specify the execute form of the macro. Use the execute form together with the list form of the macro for applications that require reentrant code. The execute form of the macro stores the

parameters into the storage area defined by the list form, and generates the macro invocation to transfer control to the service.

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

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

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

,list addr
The name of a storage area to contain the parameters. For MF=S, MF=E, and MF=M, this can be an RS-type address or an address in register (1)-(12).

,attr
An optional 1- to 60-character input string that you use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

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

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

ABEND Codes

E77
A GTZ ABEND might be issued by the system in error situations, or because it was requested via a GTZPRMxx DEBUG statement which matched the parameters of this GTZTRACK invocation request here. The ABEND reason codes are divided into two ranges to distinguish between those two situations.

In the following GTZTRACK abend reason codes, the two bytes designated by "xxxx" are for the tracker component's diagnostic purposes and have no significance to the external interface.

Reason Code (Hex)
Explanation

xxxx0rrr
This ABEND was triggered by a matching DEBUG statement, compare the GTZPRMxx parmlib member DEBUG statement or the SETGTZ DEBUG console command. The matching statement had the user specified reason code 0rrr (which must be in the range of 0000 through 0FFF).

xxxxyzzz
With y not equal to zero: Report this to the system programmer to contact IBM Service.

Return and Reason Codes

When the GTZTRACK macro returns control to your program:

- GPR 15 (and *retcode*, when you code RETCODE) contains a return code.
- When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, when you code RSNCODE) contains a reason code.

Macro GTZZTRK provides equate symbols for the return and reason codes.

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

Table 25. Return and Reason Codes for the GTZTRACK Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: GtzTrackRc_OK **Meaning**: Successful request. **Action**: None required
4	—	**Equate Symbol**: GtzTrackRc_Warn **Meaning**: Warning. **Action**: Refer to action under the individual reason code.
4	xxxx0401	**Equate Symbol**: GtzTrackRsn_NotReady **Meaning**: The tracking facility is not ready. Track data has not been recorded. This could be due the following reasons (not necessarily a complete list): • Tracking is not enabled. Compare command SETGTZ TRACKING. • The facility is in flood control mode or otherwise too busy. • The facility has not been started yet or is still starting and initialization is not complete yet. **Action**: Consider enabling tracking via command SETGTZ TRACKING=ON. Use command DISPLAY GTZ,STATUS to view additional tracking facility status. This might be a temporary condition. Compare also the description of messages GTZ1000I and GTZ0004E for further information.
8	–	**Equate Symbol**: GtzTrackRc_Error **Meaning**: Error **Action**: Refer to action under the individual reason code.
8	xxxx0801	**Equate Symbol**: GtzTrackRsn_BadGTZATRKP **Meaning**: Invalid GTZATRKP value. **Action**: Ensure that only macro GTZTRACK is used to build the parameter list for the GTZTRACK service.
8	xxxx0803	**Equate Symbol**: GtzTrackRsn_BadParmlist **Meaning**: Error accessing the GTZTRACK parameter list. **Action**: Make sure the provided parameter list is valid.
8	xxxx0804	**Equate Symbol**: GtzTrackRsn_BadSourcePath **Meaning**: Error accessing the SOURCEPATH parameter. **Action**: Make sure the provided parameter is valid.

Table 25. Return and Reason Codes for the GTZTRACK Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
8	xxxx0805	**Equate Symbol**: GtzTrackRsn_BadEventDesc **Meaning**: Error accessing the EVENTDESC parameter. **Action**: Make sure the provided parameter list is valid.
8	xxxx0806	**Equate Symbol**: GtzTrackRsn_BadEnvNotEnabled **Meaning**: Not enabled. **Action**: Avoid using GTZTRACK when not enabled.
8	xxxx0807	**Equate Symbol**: GtzTrackRsn_BadEnvLocked **Meaning**: Lock is held **Action**: Avoid using GTZTRACK when a lock is held.
8	xxxx0808	**Equate Symbol**: GtzTrackRsn_BadEnvSrbmode **Meaning**: SRB mode. **Action**: Avoid using GTZTRACK when in SRB mode.
8	xxxx0809	**Equate Symbol**: GtzTrackRsn_BadEnvFRR **Meaning**: The caller had an EUT FRR established. **Action**: Avoid using GTZTRACK when an EUT FRR is established.
8	xxxx080A	**Equate Symbol**: GtzTrackRsn_BadParmlistVersion **Meaning**: The specified version of the macro is not compatible with the current version of IBM Generic Tracker for z/OS. **Action**: Avoid requesting parameters that are not supported by this version of IBM Generic Tracker for z/OS.
8	xxxx080B	**Equate Symbol**: GtzTrackRsn_BadEventAddrZero **Meaning**: EVENTADDR is zero. **Action**: Do not specify zero as a value for the EVENTADDR. The Console Tracking facility and its service CNZTRKR used to allow for a Violator address of zero, but Generic Tracker by default does not.
8	xxxx080C	**Equate Symbol**: GtzTrackRsn_BadSourcePathLen **Meaning**: SOURCEPATHLEN is invalid. **Action**: SOURCEPATHLEN has to be between 1 and 1024.
8	xxxx080D	**Equate Symbol**: GtzTrackRsn_BadSourcePathALET **Meaning**: Bad SOURCEPATH ALET. **Action**: Make sure the ALET associated with the SOURCEPATH parameter is valid. The access register might not have been set up correctly.

Table 25. Return and Reason Codes for the GTZTRACK Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
8	xxxx080E	**Equate Symbol**: GtzTrackRsn_NotAuthorized_EventASID **Meaning**: Not authorized to specify this EVENASID value. **Action**: Only EVENTASID(HOME) can be used when not running authorized.
8	xxxx080F	**Equate Symbol**: GtzTrackRsn_BadEventASIDKey **Meaning**: Bad EVENTASID value. **Action**: Specify one of HOME, PRIMARY, SECONDARY, or VALUE.
8	xxxx0810	**Equate Symbol**: GtzTrackRsn_BadEventDescLen **Meaning**: EVENTDESCLEN is invalid. **Action**: EVENTDESCLEN has to be between 1 and 64.
8	xxxx0811	**Equate Symbol**: GtzTrackRsn_BadEventDescALET **Meaning**: Bad EVENTDESC ALET. **Action**: Make sure the ALET associated with the EVENTDESC parameter is valid. The access register might not have been set up correctly.
8	xxxx0812	**Equate Symbol**: GtzTrackRsn_NotAuthorized_CNZTRKR **Meaning**: Not authorized to specify parameter CNZTRKR. **Action**: Do not use parameter CNZTRKR and ensure that only macro GTZTRACK is used to build the parameter list for the GTZTRACK service.
8	xxxx0813	**Equate Symbol**: GtzTrackRsn_NotAuthorized_EVENTADDR0 **Meaning**: Not authorized to specify parameter EVENADDR0. **Action**: Do not use parameter EVENTADDR0 and ensure that only macro GTZTRACK is used to build the parameter list for the GTZTRACK service.
8	xxxx0814	**Equate Symbol**: GtzTrackRsn_BadParmlistALET **Meaning**: Bad parameter list ALET. **Action**: Make sure the ALET associated with the parameter list is valid. The access register might not have been set up correctly.
8	xxxx0815	**Equate Symbol**: GtzTrackRsn_BadEventASIDSlot **Meaning**: Bad EVENTASID value. **Action**: Specify one of HOME, PRIMARY, SECONDARY for EVENTASID, or specify VALUE with a valid EVENTASIDVAL.
8	xxxx0816	**Equate Symbol**: GtzTrackRsn_BadEventASID **Meaning**: Bad EVENTASID value. **Action**: Specify one of HOME, PRIMARY, SECONDARY, or VALUE, with a valid EVENTASIDVAL.

Table 25. Return and Reason Codes for the GTZTRACK Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
8	xxxx0818	**Equate Symbol**: GtzTrackRsn_BadEnvNotInGtz **Meaning**: The processing module for GTZTRACK has been invoked outside of the GTZ address space. **Action**: Use the provided GTZTRACK macro to call the processing module.
8	xxxx0819	**Equate Symbol**: GtzTrackRsn_NotAuthorized_Caller **Meaning**: A processing module for GTZTRACK has been invoked in the wrong state. **Action**: Use the provided GTZTRACK macro to call the processing module.
8	xxxx081A	**Equate Symbol**: GtzTrackRsn_BadProgramPathALET **Meaning**: Bad PROGRAMPATHALET. **Action**: Make sure the ALET associated with the PROGRAMPATH parameter is valid. The access register might not have been set up correctly.
8	xxxx081B	**Equate Symbol**: GtzTrackRsn_BadProgramPathLen **Meaning**: PROGRAMPATHLEN is invalid. **Action**: PROGRAMPATHLEN has to be between 1 and 1024.
8	xxxx081C	**Equate Symbol**: GtzTrackRsn_BadNoAbend **Meaning**: NOABEND or NOABENDVAL is invalid. **Action**: NOABEND has to be YES or NO. NOABENDVAL has to be 1 or 0.
8	xxxx081D	**Equate Symbol**: GtzTrackRsn_BadOwnerValue **Meaning**: Bad OWNER text value. **Action**: Ensure that the OWNER value contains only the documented allowed characters.
8	xxxx081E	**Equate Symbol**: GtzTrackRsn_BadEventDescValue **Meaning**: Bad EVENTDESC text value. **Action**: Ensure that the EVENTDESC value does not contain only blanks or only binary zeroes.
8	xxxx081F	**Equate Symbol**: GtzTrackRsn_BadSourceValue **Meaning**: Bad SOURCE text value. **Action**: Ensure that the SOURCE value contains only the documented allowed characters.
0D	–	**Equate Symbol**: GtzTrackRc_OutOfMemory **Meaning**: Tracking facility is low on memory. **Action**: See the description of message GTZ0004E.

Table 25. Return and Reason Codes for the GTZTRACK Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
10	–	**Equate Symbol**: GtzTrackRc_CompError **Meaning**: Component error. **Action**: Report the associated reason code to the system programmer to contact IBM Service.

Examples

None.

Chapter 103. GQSCAN — Extract information from global resource serialization queue

Description

Use the GQSCAN macro to obtain the status of resources and requestors of those resources. The GQSCAN macro allows you to obtain resource information from the system.

ISGQUERY is the IBM recommended replacement for the GQSCAN service.

The ISGRIB macro allows you to interpret the data that the GQSCAN service routine returns to the user-specified area. The ISGRIB macro maps the resource information block (RIB) and the resource information block extent (RIBE) as shown in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

There are two fields in the RIB that you can use to determine whether any RIBEs were not returned:

- RIBTRIBE contains the total number of RIBEs associated with this RIB
- RIBNRIBE contains the total number of RIBEs returned by GQSCAN with this RIB in the user-specified area indicated by the AREA parameter.

Global resource serialization counts and limits the number of outstanding global resource serialization requests. A global resource serialization request is any ENQ, RESERVE, or GQSCAN that causes an element to be inserted into a queue in the global resource serialization request queue area.

Environment

The requirements for the caller are:

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

Programming requirements

To interpret the data that the GQSCAN service routine returns in the user-specified area, you must include the ISGRIB mapping macro as a DSECT in your program.

Restrictions

Unauthorized callers of GQSCAN need to be authorized through System Authorization Facility (SAF) when Multi-level security (MLS) is active. If the caller is not authorized, the request will fail.

When multilevel security support is active on the system, unauthorized callers of ISGQUERY who specify REQINFO=QSCAN must have at least READ authorization to the ISG.QSCANSERVICES.AUTHORIZATION resource in the FACILITY class. You can activate the multilevel security support through the SETROPTS MLACTIVE option in RACF. For general information about defining profiles in the FACILITY class, see *z/OS Security Server RACF Command Language Reference* and *z/OS Security Server RACF Security Administrator's Guide*. For information about multilevel security, see *z/OS Planning for Multilevel Security and the Common Criteria*.

Input register information

Before issuing the GQSCAN macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Register 0 contains a fullword reason code if the return code in register 15 is X'0A' or X'0C'. Otherwise, register 0 contains the following two halfword values:

- The first (high-order) halfword contains the length of the fixed portion of each RIB returned.
- The second (low-order) halfword contains the length of each RIBE returned or reason code.

1
Contains the number of RIBs that were copied into the area provided

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

In general, the narrower the search parameters (particularly QNAME and RNAME), the less time it takes. Using both a specific QNAME and a specific RNAME gives better performance than using generic prefix.

The use of XSYS=YES (the default) might greatly degrade the performance of the request. Use it only when required.

Polling for ENQ contention through GQSCAN or ISGQUERY is not recommended. See the *z/OS MVS Planning: Global Resource Serialization* and *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about monitoring contention through ENF 51.

None.

Syntax

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

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede GQSCAN.
GQSCAN	
␣	One or more blanks must follow GQSCAN.
AREA=(*area addr*,*area size*)	*area addr*: A-type address or register (2) - (12).
	area size: symbol, decimal digit, or register (2) - (12). **Note:** AREA cannot be specified with QUIT=YES.
,REQLIM=*value*	*value*: symbol, decimal digit, register (2) - (12), or the word MAX.
,REQLIM=MAX	**Default**: REQLIM=MAX
,SCOPE=ALL	**Default**: SCOPE=STEP
,SCOPE=STEP	
,SCOPE=SYSTEM	
,SCOPE=SYSTEMS	
,RESERVE=YES	**Default**: All resources requested with RESERVE and all resources requested with ENQ.
,RESERVE=NO	
,RESNAME=(*qname*	*qname addr*: RX-type address or register (2) - (12).
addr[,*rname addr*,	*rname addr*: RX-type address or register (2) - (12).
rname length],	*rname length*: decimal digit, or register (2) - (12). **Default**: assembled length of *rname*.
[GENERIC|SPECIFIC],	

Syntax	Description
qname length)	**Default**: *qname length* of eight.
,SYSNAME=(*sysname addr*	*sysname addr*: RX-type address or register (2) - (12).
[,*asid value*])	*asid value*: symbol, decimal digit, or register (2) - (12). **Note:** Provide *rname addr* only when *qname addr* is used. Code *rname length* if a register is specified for *rname addr.* Code an *asid value* only when the *sysname addr* is used.
,QUIT=YES	**Default**: QUIT=NO
,QUIT=NO	**Note:** QUIT=YES is mutually exclusive with all parameters but TOKEN and MF.
,REQCNT=*value*	*value*: decimal digit or register (2) - (12).
	Default: REQCNT=0
,OWNERCT=*value*,WAITCNT=	
value	*value*: decimal digit or register (2) - (12).
,OWNERCT=*value*	*value*: decimal digit or register (2) - (12).
,WAITCNT=*value*	*value*: decimal digit or register (2) - (12).
,TOKEN=*addr*	*addr*: RX-type address or register (2) - (12).
,XSYS=YES	**Default**: XSYS=YES
,XSYS=NO	**Note:** XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.

Parameters

The parameters are explained as follows:

AREA=(*area addr*,*area size*)
Specifies the location and size of the area where information extracted from the global resource serialization resource queues is to be placed. The minimum size is the amount needed to describe a single resource, which is the length of the fixed portions of the RIB and the maximum size rname rounded up to a fullword value. IBM recommends that you use a minimum of 1024 bytes as the area size.

,REQLIM=*value*
,REQLIM=MAX
Specifies the maximum number of owners and waiters to be returned for each individual resource within the specification of RESNAME, which can be any value in the range 0 to 2^{15}–1. MAX specifies 2^{15}–1 (32767).

,SCOPE=ALL
,SCOPE=STEP
,SCOPE=SYSTEM
,SCOPE=SYSTEMS
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.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller's system.

,RESERVE=YES
,RESERVE=NO
If you specify RESERVE=YES, information is only returned for the requestors of the resource, that requested the resource with the RESERVE macro. If, for example, the resource also had requestors with the ENQ macro, the ENQ requestor's information would not be returned for the resource.

RESERVE=NO information is only returned for the requestors of the resource that requested the resource with the ENQ macro. In other words, if the resource also had requestors with the RESERVE macro, the RESERVE requestor's information would not be returned for the resource.

,RESNAME=(*qname addr*[,*rname addr*,*rname length*],[GENERIC|SPECIFIC],*qname length*)
RESNAME identifies an individual resource or group of resources that GQSCAN will examine.

RESNAME with *(rname)* indicates the name of one resource.

The *qname addr* specifies the address of the 8-character major name of the requested resource.

The *rname addr* specifies the virtual storage address of a 1 to 255-byte minor name used with the major name to represent a single resource. Information returned is for a single resource unless you specify SCOPE=ALL, in which case it could be for three resources (STEP, SYSTEM, and SYSTEMS). If the name specified by *rname* is defined by an EQU assembler instruction, the *rname length* must be specified.

The *rname length* specifies the length of the minor name. If you use the register form, specify length in the low-order (rightmost) byte. The length must match the *rname length* specified on ENQ or RESERVE.

GENERIC specifies that the *rname* of the requested resource must match but only for the length specified. For example, an ENQ for SYS1.PROCLIB would match the GQSCAN *rname* specified as SYS1 for an *rname* length of 4.

SPECIFIC specifies that the *rname* of the requested resource must exactly match the GQSCAN *rname*.

Note: GENERIC and SPECIFIC are mutually exclusive.

The *qname* length specifies the number of characters in a resource *qname* that must match the GQSCAN *qname* specified by RESNAME. You must specify a *qname* length to request a GQSCAN for a generic *qname*. For example, an ENQ with a *qname* of SYSDSN would match a GQSCAN specifying GENERIC with a *qname* of SYSD and *qname* length of 4. Specify zero for the *qname* length (with any *qname*) to request a generic GQSCAN matching any resource *qname*. If you do not specify a *qname* length, GQSCAN uses the default of 8.

,SYSNAME=(*sysname addr*[,*asid value*])
Specify SYSNAME to tell GQSCAN to return information for resources requested by tasks running on the MVS system specified in an 8-byte field pointed to by the address in *sysname address* and the *asid*

value, a 4-byte address space identifier, right justified. Valid SYSNAMEs are specified in the IEASYS*xx* parmlib member.

Information returned includes only those resources whose *sysname addr* and *asid value* match the ones specified. SYSNAME=0 or SYSNAME=(0,*asid value*), specifies that the system name is that of the system on which GQSCAN is issued. The system issues return code X'0A' with a reason code of X'0C', if SYSNAME≠0 or SYSNAME≠(0,*asid value*) is specified with XSYS=NO.

Note: Only information on resources with scope of SYSTEMS is returned from systems other than the caller's system.

,QUIT=YES
,QUIT=NO
QUIT=NO indicates that you do not want to end the current global resource serialization queue scan. QUIT=YES tells GQSCAN to stop processing the current global resource serialization queue scan and release the storage allocated to accumulate the information specified in the token.

If you specify QUIT=YES, you *must* specify the TOKEN parameter. If you specify QUIT=YES without the TOKEN parameter, the system issues abend X'09A'.

,REQCNT=*rcount*
Specifies that you want GQSCAN to return resource information only when the total number of requesters (owners plus waiters) for an individual resource is greater than or equal to *rcount*, which can be any value in the range 0 to $2^{31}-1$.

,OWNERCT=*ocount*
Specifies that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to *ocount*, which can be any value in the range 0 to $2^{31}-1$.

,WAITCNT=*wcount*
Specifies that you want GQSCAN to return resource information only when the total number of waiters for an individual resource is greater than or equal to *wcount*, which can be any value in the range 0 to $2^{31}-1$.

OWNERCT=*ocount*,WAITCNT=*wcount*
Specifies that you want GQSCAN to return resource information only when the total number of owners for an individual resource is greater than or equal to *ocount* or when the total number of waiters for an individual resource is greater than or equal to *wcount*.

,TOKEN=*addr*
Specifies the address of a fullword of storage that the GQSCAN service routine can use to provide you with any remaining information in subsequent invocations. If the token value is zero, the scan starts at the beginning of the resource queue. If the token value is not zero, the scan resumes at the point specified on TOKEN. Specify the same token value that GQSCAN returned on its previous invocation to continue where processing left off on the previous invocation.

When providing a non-zero token value, you must specify the same scope that you specified on the GQSCAN request that returned the token.

,XSYS=YES
,XSYS=NO
Specifies whether GQSCAN should be propagated across systems in the global resource serialization complex, to gather complex-wide information. This parameter is ignored in a global resource serialization ring complex, and for requests that only gather local data.

Specify XSYS=YES if the program requires complex-wide global resource serialization information. The caller might be suspended while the information is being gathered. Do *not* specify or default to XSYS=YES if this condition cannot be tolerated.

Specify XSYS=NO if the program will accept global resource serialization information from this system only. The RIBE data will contain information about requestors from other other systems in the complex only if that information is already available on the GQSCAN caller's system. Otherwise, RIBE data will be provided only for requests from the GQSCAN caller's system, and the counts in the RIB

will reflect only those requests. This request is always handled without placing the caller's dispatchable unit into a wait.

ABEND codes

See *z/OS MVS System Codes* for more information about the abend codes.

Return and reason codes

When GQSCAN returns control, register 15 contains one of the following return codes:

Table 26. Return codes for the GQSCAN macro

Hexadecimal return code	Meaning and action
00	**Meaning:** Queue scan processing is complete. Data is now in the area you specified. There is no more data to return. **Action:** Process the data.
04	**Meaning:** **Action:** **Meaning**: Queue scan processing is complete. No resources matched your request. **Action**: Do not try to process any data; none exists.
08	**Meaning:** The area you specified was filled before queue scan processing completed. **Action:** If you specified TOKEN, process the information in the area and issue GQSCAN again, specifying the TOKEN returned to you. If you did not specify TOKEN, specify a larger area or specify a TOKEN.

Table 26. Return codes for the GQSCAN macro (continued)

Hexadecimal return code	Meaning and action
0A	**Meaning:** The information you specified to GQSCAN is not valid. **Action:** Take the action indicated by the following hexadecimal reason code found in register 0. **Reason code** **Meaning** **04** The caller attempted to use GQSCAN before the global resource serialization (GRS) address space was active. **08** The size of the reply area, specified by the AREA parameter, is too small to contain a resource information block (RIB) of maximum size. **0C** You specified mutually exclusive arguments (RESERVE=YES, RESERVE=NO, RESNAME=, SYSNAME=, or XSYS=NO) to GQSCAN. **10** The caller was holding a local lock other than the GRS local lock when GQSCAN was invoked. **14** One of the following conditions, in reference to the RESNAME parameter, was detected by GQSCAN: • The *qname length* was specified with a value greater than eight. • The *qname length* value was specified without the *qname addr* value. • The SPECIFIC parameter was specified with a *rname length* value of zero. • The *rname* or *rname length* was specified without the *qname addr* value. **18** The *asid value*, for the SYSNAME parameter was specified without the *sysname addr* value. **1C** The REQCNT parameter was specified with either the OWNERCNT or WAITCNT parameters. **20** The combination of values specified on the SCOPE parameter is not valid. **28** An element in GQSCAN's input parameter list was not in the caller's storage protect key. **2C** An invalid token was specified to GQSCAN. **30** The GQSCAN caller is not authorized to use the restricted interface (SCOPE=LOCAL or GLOBAL). **34** QUIT=YES was specified without the TOKEN parameter. **38** The caller held a CMS lock other than CMSEQDQ when GQSCAN was invoked. **3C** The caller held a lock that violated the environmental restrictions of a service required by GQSCAN. **40** The caller invoked GQSCAN in the service request block (SRB) mode. **44** The value specified for the REQLIM parameter was not valid. **48** The value specified for the REQCNT parameter was not valid. **4C** The value specified for the OWNERCT parameter was not valid. **50** The value specified for the WAITCNT parameter was not valid. **54** The GQSCAN token (TOKEN) is expired. **58** SETROPTS MLACTIVE is in effect, and the program is not authorized to issue GQSCAN. Ensure the program is running authorized, or is associated with a userid with at least READ access to the best fit FACILITY class resource profile of the form ISG.QSCANSERVICES.AUTHORIZATION and that the FACILITY class is SETROPTS RACLISTed.

Table 26. Return codes for the GQSCAN macro (continued)	
Hexadecimal return code	**Meaning and action**
0C	**Meaning:** System error. Queue scan encountered an abnormal situation while processing. The information in your area is not meaningful. The reason code in register 0 contains one of the following: **Reason code** **Meaning** **00** GQSCAN has sustained an unrecoverable error. **04** The GQSCAN caller attempted to resume a scan that was started when the global resource serialization complex, which is now in star mode, was in ring mode. **08** The GQSCAN service is not able to obtain storage to satisfy the request. **0C** Sysplex processing of a SYSTEMS or GLOBAL request failed. **10** The GQSCAN service failed because the complex was migrating from a ring to a star configuration. **14** The GQSCAN service failed because inconsistent data was returned from one or more systems. **Action:** Do not try to process any data; none exists. Retry the request one or more times.
10	**Meaning:** Program error. An incorrect SYSNAME was specified as input to queue scan. The information in your area is not meaningful. **Action:** Specify a valid SYSNAME on the call to GQSCAN.
14	**Meaning:** Environmental error. The area you specified was filled before queue scan processing completed. Your request specified TOKEN, but the limit for the number of concurrent resource requests (ENQ, RESERVE, or GQSCAN) has been reached. The information in your area is valid but incomplete. The scan cannot be resumed. **Action:** Retry the request one or more times. If the problem persists, consult your system programmer, who might be able to tune the system so that the limit is no longer exceeded.

GQSCAN - List form

The list form of the GQSCAN macro is used to construct a non-executable parameter list. This parameter list, or a copy of it for reentrant programs, can be referred to by the execute form of the GQSCAN macro.

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

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GQSCAN.
GQSCAN	
ƀ	One or more blanks must follow GQSCAN.
AREA=*(area addr, area size)*	*area addr*: A-type address.
	area size: symbol, decimal digit.
	Note: 1. This parameter cannot be specified with QUIT=YES. 2. AREA is required on either the list or the execute form of the macro.

Syntax	Description
,REQLIM=*value*	*value*: symbol, decimal digit or the word MAX.
,REQLIM=MAX	**Default**: REQLIM=MAX
,SCOPE=ALL	**Default**: SCOPE=STEP
,SCOPE=STEP	
,SCOPE=SYSTEM	
,SCOPE=SYSTEMS	
,RESERVE=YES	**Default**: All resources requested with RESERVE and all
,RESERVE=NO	resources requested with ENQ.
,RESNAME=*(qname*	*qname addr*: A-type address.
addr [,rname addr,	*rname addr*: A-type address.
rname length],	*rname length*: decimal digit.
[GENERIC|SPECIFIC],	**Default**: assembled length of rname.
qname length)	**Default**: *qname length* of eight.
,SYSNAME=*(sysname addr*	*sysname addr*: A-type address.
[,asid value])	*asid value*: symbol, decimal digit.
	Note: *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr*. An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES	**Default**: QUIT=NO
,QUIT=NO	**Note:** Only TOKEN and MF=L can be specified with QUIT=YES.
,REQCNT=*value*	*value*: decimal digit. **Default**: REQCNT=0
,OWNERCT=*value*,WAITCNT=	
value	*value*: decimal digit.
,OWNERCT=*value*	*value*: decimal digit.
,WAITCNT=*value*	*value*: decimal digit.
,TOKEN=*addr*	*addr*: RX-type address.

Syntax	Description
,XSYS=YES	**Default**: XSYS=YES
,XSYS=NO	**Note:** XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.
,MF=L	

Parameters

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

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

GQSCAN - Execute form

The execute form of the GQSCAN macro can refer to and modify a remote parameter list built by the list form of the macro. There are no defaults for any of the parameters in the execute form of the macro.

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

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede GQSCAN.
GQSCAN	
ƀ	One or more blanks must follow GQSCAN.
AREA=*(area addr,area size)*	*area addr*: RX-type address or register (2) - (12).
	area size: symbol, decimal digit, or register (2) - (12).
	Note: 1. AREA cannot be specified with QUIT=YES. 2. AREA is required on either the list or the execute form of the macro.
,REQLIM=*value*	*value*: symbol, decimal digit, register (2) - (12), or the word MAX.
,REQLIM=MAX	

Syntax	Description
,SCOPE=STEP	**Note:** SCOPE=LOCAL and SCOPE=GLOBAL cannot be coded on the list form of this macro.
,SCOPE=ALL	
,SCOPE=SYSTEM	
,SCOPE=SYSTEMS	
,RESERVE=YES	
,RESERVE=NO	
,RESNAME=*(qname*	*qname addr*: RX-type address or register (2) - (12).
addr[,rname addr,	*rname addr*: RX-type address or register (2) - (12).
rname length],	*rname length*: decimal digit, register (2) - (12). **Default**: assembled length of rname.
[GENERIC|SPECIFIC],	
qname length)	
,SYSNAME=*(sysname addr*	*sysname addr*: RX-type address or register (2) - (12).
[,asid value]	*asid value*: symbol, decimal digit, or register (2) - (12).
	Note: *rname addr* can be provided only when *qname addr* is used. *rname length* must be provided if a register is specified for *rname addr.* An *asid value* can be coded only when the *sysname addr* is used.
,QUIT=YES	**Default**: QUIT=NO
,QUIT=NO	**Note:** Only TOKEN and MF=(E, *parm list addr*) can be specified with QUIT=YES.
,REQCNT=*value*	*value*: decimal digit or register (2) - (12).
	Default: REQCNT=0
,OWNERCT=*value*,WAITCNT= *value*	*value*: decimal digit.
,OWNERCT=*value*	*value*: decimal digit.
,WAITCNT=*value*	*value*: decimal digit.
,TOKEN=*addr*	*addr*: RX-type address of register (2) - (12).

Syntax	Description
,XSYS=YES	**Default**: XSYS=YES
,XSYS=NO	**Note:** XSYS=NO is mutually exclusive with TOKEN, QUIT=YES and SYSNAME, when SYSNAME is not equal to zero or zero and the asid value(0,asid value). In a global resource serialization ring complex, XSYS=NO is ignored.
,MF=(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 104. 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:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN **Note:** PASN=HASN=SASN is required for a nonshared standard hiperspace for which an ALET is not used (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.

Input register information

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

However, if the caller specifies the HSPALET parameter:

- General purpose register (GPR) 13 must contain the address of a 144-byte save area. The save area must be in the caller's primary address space.
- Access register (AR) 13 must contain 0, regardless of whether the caller is in primary or AR address space control (ASC) mode.

Output register information

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

Register
Contents

0
Reason code

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

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

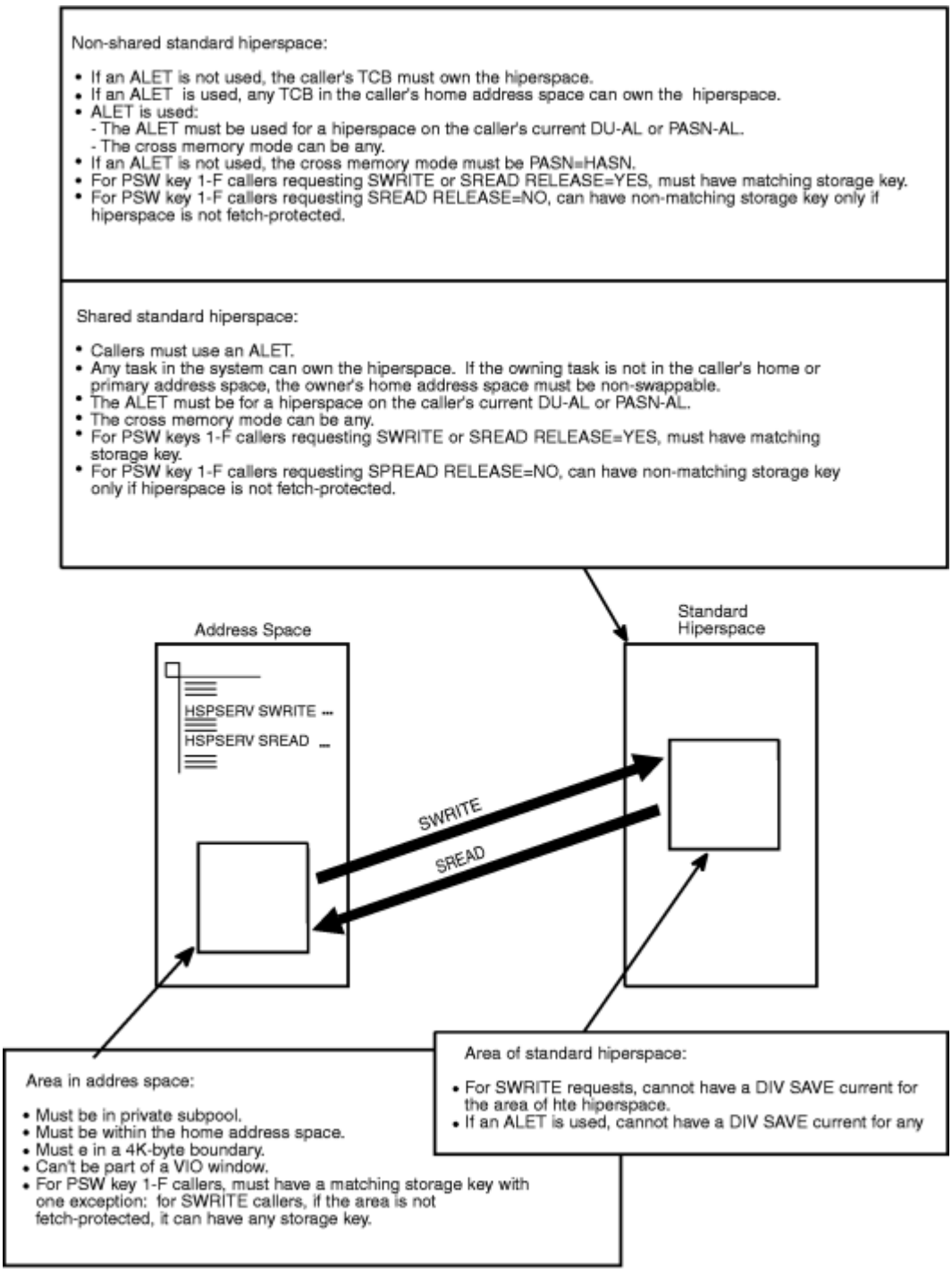


Figure 10. Characteristics and Restrictions for Standard Hiperspaces

Syntax

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

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

Syntax	Description
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	
ƀ	One or more blanks must follow HSPSERV.
SREAD SWRITE	
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=*n*	*n*: Number from 1 to 50.
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12).
	Default: NUMRANGE=1.
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - 12).
,RELEASE=NO	**Default**: RELEASE=NO.
,RELEASE=YES	
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

SREAD
Requests that the system read data from a standard hiperspace to an address space.

STOKEN and RANGLIST are required parameters on the SREAD request. 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 HSPSERV returns to the caller after the SWRITE operation, the contents of the address space storage range are not preserved. You can use the address space area again.
- If the hiperspace maps a data-in-virtual object, do not issue an SWRITE request while a DIV SAVE request is current.

STOKEN and RANGLIST are required parameters on the SWRITE request. 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 10 on page 709.

,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:

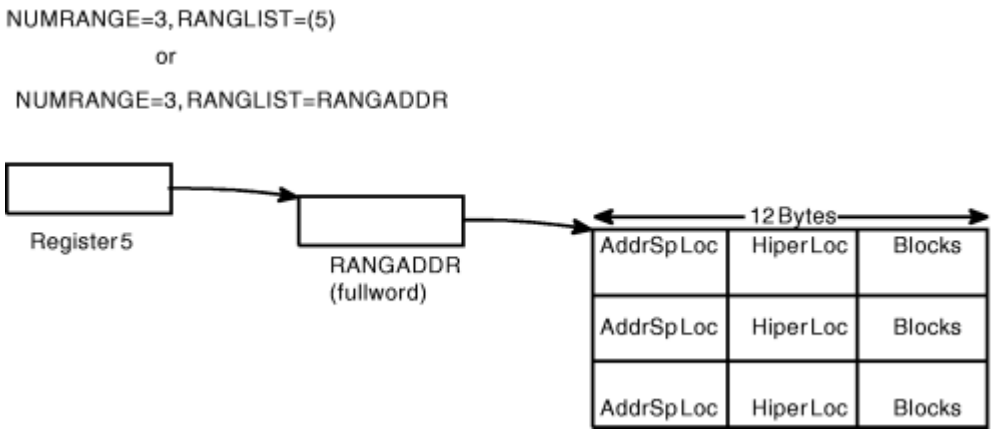
First Word
The starting virtual address in the address space into which the data is to be read or from which the data is to be written

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 10 on page 709.

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.

,RSNCODE=*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'.

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.

Return Code	Reason Code	Meaning and Action
00	00	**Meaning**: HSPSERV completed successfully. **Action**: None.

Return Code	Reason Code	Meaning and Action
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:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	
ƀ	One or more blanks must follow HSPSERV.
PLISTVER=*vernum*	*vernum*: Parameter list version 0 or 1. **Default**: Version that allows all specified parameters.
,MF=(L,*list addr*)	*list addr*: Symbol.
,MF=(L,*list addr,attr)*	*attr*: 1- to 60-character input string. **Default**: 0D

Parameters

Parameters for the list form of HSPSERV are as follows:

PLISTVER=*vernum*
Specifies the macro version associated with HSPSERV. PLISTVER is an optional parameter that determines which parameter list the system generates. Specify zero if you use parameters only from this group:

- MF
- NUMRANGE
- PLISTVER
- RANGLIST
- RELEASE
- RETCODE
- RSNCODE
- SREAD
- STOKEN
- SWRITE

If you use the HSPALET parameter, specify 1.

If you do not specify PLISTVER, the default is to allow all of the parameters you specify on the invocation to be processed.

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

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

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

HSPSERV - Execute form

The execute form of the HSPSERV macro changes parameters in the control parameter list that the system created through the list form of the macro and performs the specified operation.

Syntax

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

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

Syntax	Description
ƀ	One or more blanks must follow HSPSERV.
SREAD SWRITE	
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=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 HSPSERV macro with the following exceptions:

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

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

COMPLETE, which is the default, specifies that the system 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.

HSPSERV - Modify form

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

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede HSPSERV.
HSPSERV	
ƀ	One or more blanks must follow HSPSERV.
SREAD SWRITE	
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,HSPALET=*alet-addr*	*alet-addr*: RX-type address or register (2) - (12).
,NUMRANGE=1	**Default**: NUMRANGE=1.
,NUMRANGE=*num-addr*	*num-addr*: RX-type address or register (2) - (12).
,RANGLIST=*list-addr*	*list-addr*: RX-type address or register (2) - (12).
,RELEASE=NO	**Default**: RELEASE=NO.
,RELEASE=YES	
,RETCODE=*ret-addr*	*ret-addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn-addr*	*rsn-addr*: RX-type address or register (2) - (12).

Syntax	Description
,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,*list-addr*,COMPLETE)
,MF=(M,*list-addr*,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 A. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed message to the Contact the z/OS team web page (www.ibm.com/systems/campaignmail/z/zos/contact_z) or use the following mailing address.

IBM Corporation
Attention: MHVRCFS Reader Comments
Department H6MA, Building 707
2455 South Road
Poughkeepsie, NY 12601-5400
United States

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or 3.1 or 3.1.1. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number 3.1)

are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The `*` symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element `*FILE` with dotted decimal number 3 is given the format `3 \* FILE`. Format `3* FILE` indicates that syntax element `FILE` repeats. Format `3* \* FILE` indicates that syntax element `* FILE` repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines `5.1*, 5.1 LASTRUN, and 5.1 DELETE` mean that if you use more than one of the `LASTRUN` and `DELETE` syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line `2.1 %OP1` means that you must refer to separate syntax fragment `OP1`.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example `5? NOTIFY`). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines `5 ?, 5 NOTIFY`, and `5 UPDATE`, you know that the syntax elements `NOTIFY` and `UPDATE` are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines `2? FILE, 2.1! (KEEP), and 2.1 (DELETE)`, you know that `(KEEP)` is the default option for the `FILE` keyword. In the example, if you include the `FILE` keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the `FILE` keyword is omitted, the default `FILE(KEEP)` is used. However, if you hear the lines `2? FILE, 2.1, 2.1.1! (KEEP)`, and `2.1.1 (DELETE)`, the default option KEEP applies only to the next higher dotted decimal number, `2.1` (which does not have an associated keyword), and does not apply to `2? FILE`. Nothing is used if the keyword `FILE` is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line `5.1* data area`, you know that you can include one data area, more than one data area, or no data area. If you hear the lines `3* , 3 HOST, 3 STATE`, you know that you can include `HOST, STATE`, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.
3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line 6.1+ data area, you must include at least one data area. If you hear the lines 2+, 2 HOST, and 2 STATE, you know that you must include HOST, STATE, or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the Knowledge Centers. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMSdfp, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those

products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Programming interface information

This information is intended to help the customer to code macros that are available to all assembler language programs. This information documents intended programming interfaces that allow the customer to write programs to obtain services of z/OS.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

UNIX is a registered trademark of The Open Group in the United States and other countries.

Index

Numerics

A

B

C

D

E

F

G

H

K

L

M

N

O

P

R

S

T

U

V

X

Z

Product Number: 5650-ZOS

SA23-1369-40