z/OS
2.4

MVS Programming: Authorized Assembler Services Reference, Volume 1 (ALE-DYN)

SA23-1372-40

Note

Before using this information and the product it supports, read the information in “Notices” on page 595.

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: 2021-06-21

Contents

Figures

Tables

About this information

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

- Running in supervisor state
- Running under PSW key 0-7
- Running with APF-authorization.

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

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

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

Who should use this information

This information is for the programmer who is using assembler language to code a system program. A system program is usually one that runs in supervisor state or runs with PSW key 0-7 or runs with APF authorization.

The information assumes a knowledge of the computer, as described in *Principles of Operation*, as well as an in-depth 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 Documentation (www.ibm.com/docs/en/hla-and-tf/1.6).

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 Documentation (www.ibm.com/docs/en/zos).

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 xxxi.

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® Documentation function
If your comment or question is about the IBM Documentation functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Documentation Support at ibmdocs@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 Authorized Assembler Services Reference ALE-DYN, SA23-1372-50
- 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

The following content is new, changed, or no longer included in V2R4.

New

The following new information is added in this publication:

Prior to June 2020 refresh

- The CSVDYLPA macro is updated to add the VOLUME parameter. For more information, see "REQUEST=ADD option of CSVDYLPA" on page 326.

Changed

The following information is changed in this publication:

May 2021 refresh

- The SCOPE and CALLERKEY parameter descriptions of the DSPSERV macro are updated in "Description" on page 555.

April 2021 refresh

- The syntax and description of the CALLRTM macro are updated in Chapter 17, "CALLRTM — Call recovery termination manager," on page 151.

March 2021 refresh

- The KEY parameter is updated in "ATTACH and ATTACHX description" on page 67.
- The description of the BPXESMF macro is updated in "Description" on page 141.

June 2020 refresh

- For APAR OA56132, clarification is provided regarding the use of the MULTIHDR=YES keyword with various services in Chapter 33, "CPOOL — Perform cell pool services," on page 257.

Prior to June 2020 refresh

- The ALESERV macro is updated to add the optional keyword REFTYPE. For more information, see "Description" on page 27.
- The AXREXX macro is updated with return code OCx and reason code xxxx0C12. For more information, see "Return and reason codes" on page 116.

Summary of changes for z/OS Version 2 Release 3

The following information is new, changed, or deleted in z/OS Version 2 Release 3 (V2R3).

New

The following information has been added:

- Revised the explanation of the APFREQUIRED parameter for "REQUEST=ADD option of CSVDYLPA" on page 326.

- Added new ON parameter to CTRACE “Description” on page 479.
- Added new AC1 and LONGPARMOK parameters to CSVDYLPA “REQUEST=ADD option of CSVDYLPA” on page 326.
- New reason codes have been added to the DIV service in Table 55 on page 538.

Changed

The following information has been changed:

- Information about the ASCRE macro has been updated for the TRMEXIT parameter in Chapter 3, “ASCRE — Create address spaces,” on page 43.
- The ,ADDRENV=SAME description of the ATTACHX macro was enhanced for clarity.
- Information is corrected in the description in Example 3 of the Chapter 17, “CALLRTM — Call recovery termination manager,” on page 151.
- Information was added for the RSAPF parameter in the description for “ATTACH and ATTACHX description” on page 67.

Summary of changes for z/OS Version 2 Release 2

The following information is new, changed, or deleted in z/OS Version 2 Release 2 (V2R2).

New

The following information has been added:

- Several new parameters to support contractible multi-header cell pools have been added in Chapter 33, “CPOOL — Perform cell pool services,” on page 257.
- Several new parameters (DELETEFORCE, DISABLEDCALL, SERVICEID, SERVICEMASK) and a new parameter value (EXAAVER=3) have been added in Chapter 38, “CSVDYNEX — Provide dynamic exits services,” on page 357.

Changed

The following information has been changed:

- Updates have been made to the Description section in Chapter 7, “ATTACH and ATTACHX — Create a subtask,” on page 67.
- The description of the SVAREA parameter has been updated in Chapter 19, “CIRB - Create interruption request block,” on page 163.
- Updates have been made to the Parameters section in Chapter 46, “DOM — Delete operator message,” on page 549.

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: Extended Addressability Guide*.

Address space control (ASC) mode

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

Some macros can generate code that is appropriate for programs in either primary mode or AR mode. These macros check a global symbol set by the SYSSTATE macro. See Telling the system about the execution environment 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
- An ALET for a private entry on the PASN-AL or the DU-AL

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	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 macro, 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.

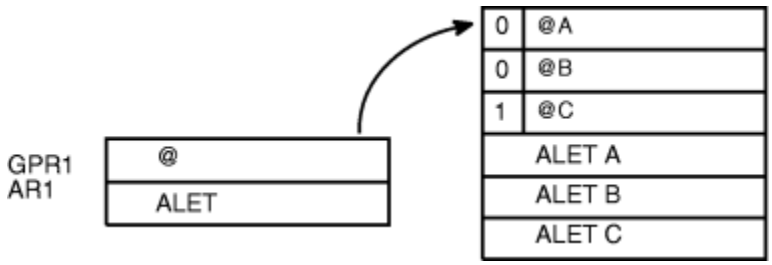

Figure 1. Sample User Parameter List for Callers in AR Mode

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: Authorized Assembler Services Guide* for information about writing recovery routines.

Handling environmental and system errors

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

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

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

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

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

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

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

Using X-macros

Some MVS services support callers in both primary and AR ASC mode. When the caller is in AR mode, macros must generate larger parameter lists; the increased size of the list reflects the addition of ALETs to qualify addresses, as described under ALET qualification. 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 authorized X-macros are:

- ATTACHX
- ESTAEX
- SDUMPX
- SYNCHX

The only way these macros know that a caller is in AR mode is by checking the global symbol that the SYSSTATE macro sets. Each of these macros (and corresponding non-X-macro) checks the symbol. If SYSSTATE ASCENV=AR has been issued, the macro issues code that is valid for callers in AR mode. If it

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

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

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

- Callers in primary mode can invoke either macro.

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

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

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

Macro forms

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

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

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

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

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 Documentation (www.ibm.com/docs/en/hla-and-tf/1.6).

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.

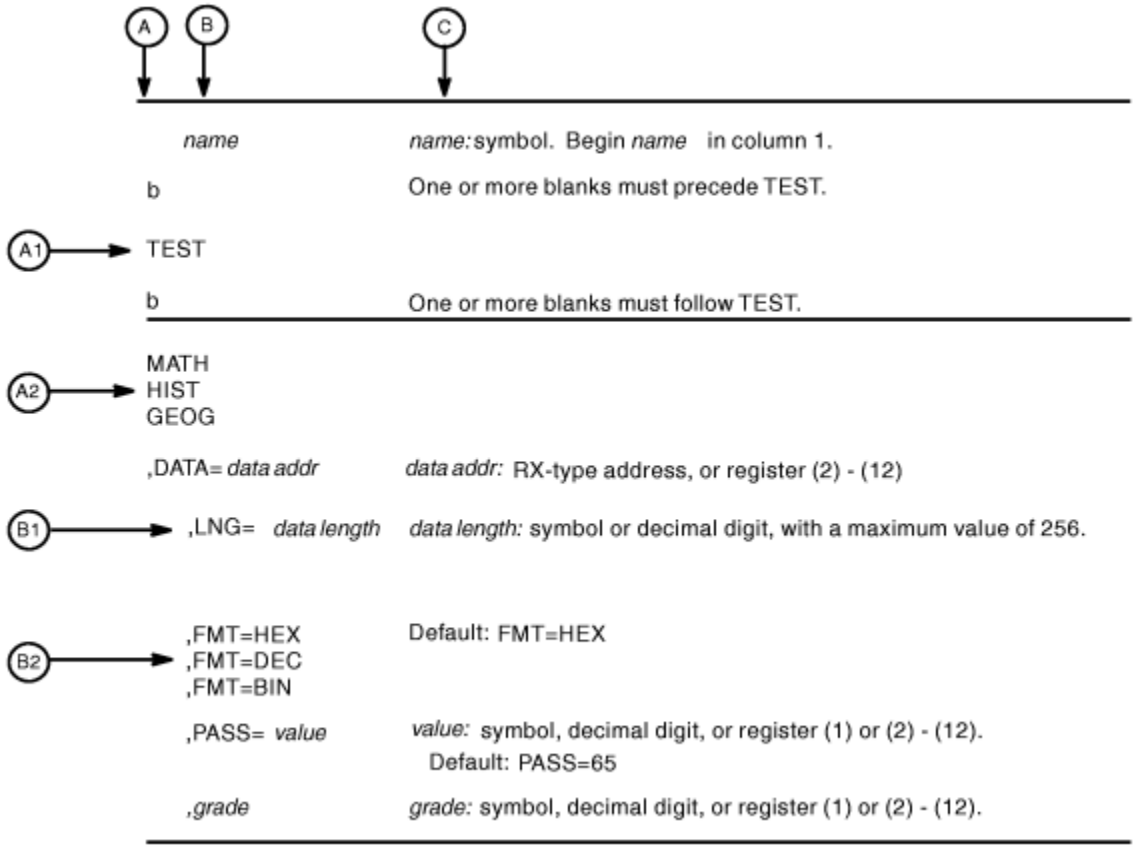

Figure 2. Sample tabular syntax diagram for the TEST macro

- 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.

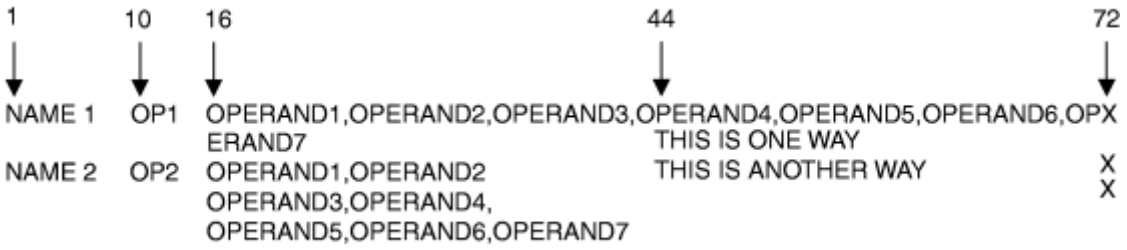

Figure 3. Continuation Coding

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: 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*.

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
ALESERV	Yes	Yes	No	No
ASCRE	Yes	Yes	Yes	No
ASDES	Yes	Yes	Yes	No
ASEXT	Yes	Yes	No	No
ATSET	No	Yes	Yes	No
ATTACH	Yes (See note 1)	No	Yes	No
ATTACHX	Yes	No	Yes	Yes
AXEXT	No	Yes	Yes	No
AXFRE	No	Yes	Yes	No
AXRES	No	Yes	Yes	No
AXREXX	No	Yes	Yes	Yes
AXSET	No	Yes	Yes	No
BPXEKDA	Yes	No	Yes	No
BPXESMF	Yes	No	Yes	No
CALLDISP	No	Yes	No	Yes
CALLRTM	No	Yes (See note 2)	No	No
CHANGKEY	No	Yes	No	No
CIRB	No	No	No	No
CMDAUTH	No	No	No	No
CNZMXURF	No	Yes	No	No
CNZTRKR	No	Yes	No	No
COFCREAT	Yes	Yes	Yes	No
COFDEFIN	Yes	Yes	Yes	No
COFIDENT	Yes	Yes	Yes	No
COFNOTIF	Yes	Yes	Yes	No
COFPURGE	Yes	Yes	Yes	No
COFREMOV	Yes	Yes	Yes	No
COFRETRI	Yes	Yes	Yes	No
COFSDONO	No	No	Yes	No
CONFCHG	No	No	Yes	No

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
CPF	No	No	No	No
CPOOL	No	Yes	Yes	No
CPUTIMER	No	Yes	Yes	No
CSRSI	No	Yes	No	No
CSRUNIC	Yes	Yes	No	No
CSVAPF	Yes (See note 11)	Yes (See note 12)	Yes	No
CSVDYNEX	Yes (See note 13)	Yes (See note 14)	Yes	No
CTRACE	No	No	Yes	No
CTRACECS	Yes	No	Yes	No
CTRACEWR	Yes	Yes	Yes	No
DATOFF	Yes	No	No	No
DEQ	No	Yes	Yes	Yes
DIV	Yes	No	Yes	No
DOM	No	No	No	Yes
DSPSERV	Yes	Yes	Yes	Yes
DYNALLOC	No	No	No	Yes
EDTINFO	No	Yes	Yes	Yes
ENFREQ	No	No	No	No
ENQ	No	Yes	Yes	Yes
ESPIE	No	No	No	Yes
ESTAE (See note 3)	No	No	Yes	No
ESTAEX	Yes	Yes	Yes	Yes
ETCON	No	Yes	Yes	No
ETCRE	No	Yes	Yes	No
ETDEF	Yes	Yes	No	No
ETDES	No	Yes	Yes	No
ETDIS	No	Yes	Yes	No
EVENTS	No	No	No	No
EXTRACT	No	No	No	No
FESTAE	No	No	No	No
FREEMAIN	Yes (See note 4)	Yes	Yes	Yes
GETDSAB	No	No	Yes	No
GETMAIN	Yes (See note 4)	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**
GQSCAN	No	Yes	No	No
GTRACE	No	Yes	No	Yes
HSPSERV	Yes	Yes (See note 5)	(See note 6)	No
IARCP64	Yes	Yes	Yes	Yes
IARR2V	Yes	Yes	No	No
IARSUBSP	Yes	Yes	Yes	No
IARST64	Yes	Yes	Yes	Yes
IARVSERV	Yes	Yes	Yes	No
IARV64	Yes	Yes	Yes	Yes
IAZXCTKN	Yes	Yes	Yes	No
IAZXJSAB	Yes	Yes (See note 15)	Yes	No
IEAARR	Yes	Yes	Yes	Yes
IEAFP	Yes	Yes	Yes	No
IEALSQRY	Yes	Yes	Yes	No
IEAMETR	Yes	Yes	Yes	No
IEAMRMF3	No	Yes	No	No
IEAMSCHD	Yes	Yes	Yes	No
IEANTCR	Yes	Yes	N/A	No
IEANTDL	Yes	Yes	N/A	No
IEANTRT	Yes	Yes	N/A	No
IEARBUP	Yes	Yes	Yes	No
IEATDUMP	Yes	No	Yes	No
IEATEDS	Yes	Yes	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
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

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
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
IEECMDS	Yes	Yes	Yes	No
IEEQEMCS	Yes	Yes	Yes	No
IEEVARYD	No	No	Yes	No
IEFPPSCN	No	No	Yes	No
IEFQMREQ	No	No	No	No
IEFSSI	Yes	No	No	No
IEFSSVT	Yes	No	No	No
IEFSSVTI	Yes	Yes	No	No
IFAQUERY	Yes	Yes	No	No
IOCINFO	Yes	Yes	No	No
IOSADMF	No	No	Yes	No
IOSCAPF	No	Yes (See note 7)	Yes	No
IOSCAPU	Yes	Yes (See note 7)	Yes	No
IOSCDR	No	No	Yes	No
IOSCHPD	Yes	Yes	Yes	No
IOSCMXA	No	Yes (See note 7)	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**
IOSCMXR	No	Yes (See note 7)	Yes	No
IOSDCXR	No	Yes (See note 7)	Yes	No
IOSENQ	Yes	Yes	Yes	No
IOSINFO	No	No	No	No
IOSLOOK	No	No	No	No
IOSPTHV	No	No	Yes	No
IOSSPOF	No	Yes	Yes	Yes
IOSUPFA	No	Yes	Yes	No
IOSUPFR	No	Yes	Yes	No
IOSVRYSW	Yes	Yes	Yes	No
IOSWITCH	Yes	Yes	Yes	No
IOSZHPF	Yes	Yes	Yes	No
IRDFSD	Yes	Yes	Yes	No
IRDFSDU	Yes	Yes	Yes	No
ISGADMIN	Yes	Yes	Yes	Yes
ISGECA	Yes	Yes	Yes	Yes
ISGENQ	Yes	Yes	Yes	Yes
ISGLCRT (See note 16)	No	Yes	N/A	No
ISGLID (See note 16)	No	Yes	N/A	Yes
ISGLOBT	No	Yes	N/A	No
ISGLREL	No	Yes	N/A	No
ISGLPRG	No	Yes	N/A	No
ISGQUERY	Yes	Yes	Yes	Yes
ITTFMTB	No	No	No	No
ITZXFILT	No	Yes	Yes	No
IWMCLSFY	No	Yes	Yes	No
IWMCONN	No	Yes	Yes	No
IWMDISC	No	Yes	Yes	No
IWMECQRY	No	Yes	Yes	No
IWMECREA	No	Yes	Yes	No
IWMEDELE	No	Yes	Yes	No
IWMMABNL	No	Yes	No	No

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**
IWMMCHST	No	Yes	No	No
IWMMCREA	No	Yes	Yes	No
IWMMDELE	No	Yes	Yes	No
IWMMEXTR	No	Yes	Yes	No
IWMMINIT	No	Yes	No	No
IWMMNTFY	No	Yes	Yes	No
IWMMRELA	No	Yes	Yes	No
IWMMSWCH	No	Yes	Yes	No
IWMMXFER	No	Yes	No	No
IWMPQRY	Yes	Yes	Yes	No
IWMRCOLL	Yes	Yes	Yes	No
IWMRPT	No	Yes	Yes	No
IWMRQRY	Yes	Yes	Yes	No
IWMSRDRS	No	Yes	Yes	No
IWMSRSRG	No	Yes	Yes	No
IWMSRSRS	No	Yes	Yes	No
IWMWMCON	No	Yes	Yes	No
IWMWQRY	Yes	Yes	Yes	No
IWMWQWRK	No	Yes	Yes	No
IXCCREAT	Yes	Yes	Yes	No
IXCDELET	Yes	Yes	Yes	No
IXCJOIN	Yes	No	Yes	No
IXCLEAVE	Yes	No	Yes	No
IXCMG	Yes	Yes	Yes	No
IXCMOD	Yes	Yes	Yes	No
IXCMSGI	Yes	No	Yes	No
IXCMSGO	Yes	Yes	Yes	No
IXCQUERY	Yes	Yes	Yes	No
IXCQUIES	Yes	No	Yes	No
IXCSETUS	Yes	Yes	Yes	No
IXCTERM	Yes	Yes	Yes	No
IXGBRWSE	Yes	Yes	Yes	Yes
IXGCONN	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**
IXGDELET	Yes	Yes	Yes	Yes
IXGWRITE	Yes	Yes	Yes	Yes
LLACOPY	No	No	Yes	No
LOAD	Yes	No	No	Yes
LOADWAIT	No	Yes	Yes	No
LOCASCB	Yes	Yes	Yes	No
LXFRE	No	Yes	Yes	No
LXRES	No	Yes	Yes	No
MCSOPER	Yes	No	Yes	No
MCSOPMSG	Yes	No	Yes	No
MGCR	No	No	No	No
MGCRE	No	No	No	No
MIHQUERY	Yes	No	Yes	No
MODESET	No	Yes	No	Yes
NIL	Yes	Yes	Yes	No
NMLDEF	No	No	No	No
NUCLKUP	No	No	No	No
OIL	Yes	Yes	Yes	No
OUTADD	No	No	No	No
OUTDEL	No	No	No	No
PCLINK	No	Yes	No	No
PGANY	No	No	No	No
PGFIX	No	Yes	No	No
PGFIXA	No	No	No	No
PGFREE	No	Yes	No	No
PGFREEA	No	No	No	No
PGSER	Yes (See note 8)	Yes (See note 8)	No	Yes
POST	No	Yes	No	Yes
PTRACE	No	Yes	No	No
PURGEDQ	No	No	No	No
QEDIT	No	No	No	No
RESERVE	No	No	No	Yes
RESMGR	Yes	Yes	No	No

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
RESUME	No	Yes	No	No
RISGNL	No	Yes	No	No
SCHEDIRB	Yes	No	Yes	No
SCHEDULE	Yes	Yes	Yes	No
SCHEDXIT	No	Yes	No	No
SDUMP	Yes (See note 1)	Yes (See note 9)	Yes	No
SDUMPX	Yes	Yes (See note 9)	Yes	Yes
SETFRR	Yes	Yes	Yes	No
SETLOCK	Yes	Yes	Yes	No
SETRP	Yes	Yes	Yes	Yes
SJFREQ	No	Yes	No	No
SPIE	No	No	No	No
SPOST	No	No	No	No
SRBSTAT	No	Yes	No	No
SRBTIMER	No	No	No	No
STATUS	Yes	Yes	No	No
STORAGE	Yes	Yes	No	Yes
SUSPEND	No	Yes	No	No
SVCUPDTE	No	No	No	No
SWAREQ	No	No	No	No
SWBTUREQ	No	No	No	No
SYMREC	No	Yes	Yes	No
SYNCH	Yes (See note 1)	No	Yes	No
SYNCHX	Yes	No	Yes	Yes
SYSEVENT	No	No	No	No
TCBTOKEN	Yes	Yes	No	No
TCTL	No	No	No	No
TESTAUTH	No	No	No	No
TIMEUSED	Yes (See note 10)	Yes	No	Yes
T6EXIT	No	No	No	No
UCBINFO	Yes	Yes	Yes	No
UCBLOOK	Yes	Yes	Yes	No
UCBPIN	Yes	Yes	Yes	No

Table 3. Service Summary (continued)				
Service	**Can be issued in AR ASC mode**	**Can be issued in cross memory mode**	**Checks SYSSTATE**	**Can be issued in 64-bit AMODE**
UCBSCAN	Yes	Yes	Yes	No
VSMLIST	No	Yes	Yes	No
VSMLOC	No	Yes	Yes	No
VSMREGN	No	Yes	No	No
WAIT	No	Yes	No	Yes
WTL	No	No	No	No
WTO	No	No	No	Yes
WTOR	No	No	No	Yes

Notes:

1. Primary mode callers can use either macro in the following macro pairs:
   - ATTACH or ATTACHX
   - SDUMP or SDUMPX
   - SYNCH or SYNCHX

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

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

12. For CSVAPF with the ADD, DELETE, and DYNFORMAT requests, PASN = HASN = SASN. For CSVAPF with the QUERY, QUERYFORMAT, and LIST requests, any PASN, any HASN, any SASN.
13. For a QUERY or a CALL request with FASTPATH=YES, CSVDYNEX can be issued only in primary mode. For all other requests, CSVDYNEX can be issued in primary or AR mode.
14. For CSVDYNEX CALL, RECOVER, and QUERY requests, any PASN, any HASN, any SASN. For all other requests, PASN=HASN=SASN.
15. When the caller of the IAZXJSAB macro specifies the ASCB parameter, any PASN, any HASN, any SASN; otherwise, PASN=HASN is required.
16. The 64 bit entry names are as follows:
    - ISGLCR64
    - ISGLID64
    - ISGLOB64
    - ISGLRE64
    - ISGLPB64
    - ISGLPR64

Chapter 2. 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, a data space, a subspace, or a hiperspace to which programs have access. Access list entry tokens (ALETs) index the entries in the access list.

On the ALESERV macro, address spaces, data spaces, subspaces, and hiperspaces are identified through STOKENs, an identifier similar to an address space identifier (ASID). *z/OS MVS Programming: Extended Addressability Guide* describes STOKENs, ALETs and how to pass them, access lists, and the EAX-checking that might occur when you issue the ALESERV macro to add an entry for an address space.

You access data spaces and address spaces, and reference subspaces, through assembler instructions. You access hiperspace through the HSPSERV macro.

Use the ALESERV macro to:

- Add an entry to a DU-AL or PASN-AL for a SCOPE=SINGLE data space, a SCOPE=ALL data space, or a hiperspace (ADD parameter)
- Add an entry to a DU-AL for a subspace (ADD parameter)
- Add an entry to all PASN-ALs for a SCOPE=COMMON data space (ADD parameter)
- Add the primary address space to the DU-AL (ADDPASN parameter)
- Delete an entry from a DU-AL or PASN-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).

ALESERV is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, except for the CHKEAX parameter.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state, with any PSW key. To request the following ALESERV services, the program must be supervisor state or PSW key 0 - 7: • Make ADD and DELETE requests for SCOPE=ALL and SCOPE=COMMON data spaces, shared standard, and extended storage only (ESO) hiperspace for the PASN-AL. • Use the CHKEAX=NO parameter. • Make ADD and DELETE requests for SCOPE=ALL and SCOPE=COMMON data spaces and shared hiperspace and ESO hiperspace for the DU-AL.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31- bit

Environmental factor	Requirement
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 is not required to hold locks.
Control parameters:	Must exist in an addressable area

Programming requirements

To add a subspace entry to a DU-AL, the caller must be running under the task that created the subspace.

Restrictions

None.

Input register information

Before issuing the ALESERV macro, the caller does not have to place information into any register unless they use it in register notation for a particular parameter, or use 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 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 (ARs) 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 to remain the same before and after they issue a service. If the system changes the contents of registers on which the caller depends, the caller must save them before they issue 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, CHKEAX, CHKPT, REFTYPE, RELATED
ADDPASN	ALET, CHKPT, RELATED
DELETE	ALET, CHKEAX, RELATED
EXTRACT	ALET, STOKEN, RELATED
SEARCH	AL, ALET, STOKEN, 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.

Syntax	Description
,CHKEAX=YES	**Default:** CHKEAX=YES
,CHKEAX=NO	
,CHKPT=FAIL	**Default:** CHKPT=FAIL
,CHKPT=IGNORE	
,REFTYPE=ANY	**Default:** REFTYPE=ANY
,REFTYPE=FETCHONLY	
,RELATED=*any-value*	*any-value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

ADD

Requests that the system add an entry to the access list and return the ALET. You are required to use two parameters:

- STOKEN specifies the space for which the entry is to be added
- ALET specifies the address of the location where the system returns the ALET.

You can also specify whether the access list is DU-AL or PASN-AL (AL parameter), for address spaces, whether the entry is PUBLIC or PRIVATE (ACCESS parameter). The defaults are DU-AL and PUBLIC.

A subspace access list entry must be added to the DU-AL as a public entry.

To add an entry for a SCOPE=COMMON data space to all PASN-ALs in the system, use the AL=PASN parameter on ALESERV ADD.

To add an entry for an address space, the problem state, PSW key 8 - F caller must have EAX-authority to the target address space. The supervisor state or PSW key 0 - 7 caller can use the CHKEAX=NO parameter, which adds an entry for the address space without requiring the caller to have EAX-authority.

To ensure the integrity of data spaces and hiperspace, the system has certain rules for adding entries for data spaces and hiperspace to access lists. The following two tables summarize the rules for problem state programs with PSW key 8 - F and supervisor state or PSW key 0 - 7 programs.

Table 4. Rules for Adding Entries for Data Spaces to Access Lists

Function	Type of data space	A problem state program with PSW key 8 - F:	A supervisor state or key 0-7 program:
Add entries to the DU-AL	SCOPE=SINGLE	Can add entries for the data spaces it owns or creates.	Can add entries if the caller's home and owner's home address space is the same.
Add entries to the DU-AL	SCOPE=ALL and SCOPE=COMMON	Cannot add entries.	Can add entries.

Table 4. Rules for Adding Entries for Data Spaces to Access Lists (continued)			
Function	**Type of data space**	**A problem state program with PSW key 8 - F:**	**A supervisor state or key 0-7 program:**
Add entries to the PASN-AL	SCOPE=SINGLE	Can add entries if the caller owns or creates the data space and the data space is not already on the PASN-AL through the actions of a problem state program with PSW key 8 - F.	Can add entries if PASN-AL is the same as the PASN-AL of the owner's home address space.
Add entries to the PASN-AL	SCOPE=ALL and SCOPE=COMMON	Cannot add entries.	Can add entries for SCOPE=COMMON data spaces. Can add entries for SCOPE=ALL data spaces if no unauthorized program can run in the primary address space.

Table 5. Rules for Adding Entries for Hiperspace to Access Lists			
Function	**Type of hiperspace**	**A problem state program with PSW key 8 - F:**	**A supervisor state or key 0-7 program:**
Add entries to the DU-AL	Nonshared standard	Can add entries for the hiperspaces it owns.	Can add entries if the caller's home and owner's home address space is the same.
Add entries to the DU-AL	Shared standard and ESO	Cannot add entries.	Can add entries.
Add entries to the PASN-AL	Nonshared standard	Cannot add entries.	Can add entries if PASN-AL is the same as the PASN-AL of the owner's home address space.
Add entries to the PASN-AL	Shared standard and ESO	Cannot add entries.	Can add entries for shared standard hiperspace. Can add entries for ESO hiperspace if no unauthorized program can run in the primary address space.

An access list entry for an ESO hiperspace might not be available to an unauthorized program.

The following notes are for users of data-in-virtual and hiperspaces.

- Once you add an entry for a standard hiperspace, you cannot use that hiperspace as a data-in-virtual object
- If a DIV ACCESS is in effect for a standard hiperspace, you cannot add an entry for that hiperspace.

ADDPASN
Requests that the system add the primary address space to the DU-AL without requiring a user to have EAX-authority to the address space. The entry is a public entry. ALET, required with ADDPASN, receives the ALET that identifies the entry.

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

To delete an entry for an address space, the problem state program with PSW key 8 - F must have EAX-authority to the target address space. The supervisor state or PSW key 0 - 7 caller can use the CHKEAX=NO parameter, which deletes an entry for the address space without requiring the caller to have EAX-authority.

When the request is for a SCOPE=COMMON data space, ALESERV deletes the entry from all PASN-ALs in the system.

EXTRACT
Requests that the system find the STOKEN associated with the specified ALET. The caller can obtain the STOKEN for any space that is represented by a valid entry on the current access list. STOKEN is a required parameter.

SEARCH
Requests that the system search through the DU-AL or PASN-AL for an ALET that corresponds to a specified STOKEN. Specify whether the search is to be through the DU-AL or the PASN-AL. (AL=WORKUNIT is the default.) ALET and STOKEN are required parameters.

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, subspace, or hiperspace. The default is ACCESS=PUBLIC.

,AL=WORKUNIT
,AL=PASN
Specifies whether the access list is a DU-AL (WORKUNIT) or a PASN-AL (PASN). The default is AL=WORKUNIT.

For the ADD request, AL identifies the type of access list. To add entries for data spaces and hiperspace to the DU-AL and PASN-AL, see the rules described in Table 4 on page 30 and Table 5 on page 31.

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

When you add or search for a subspace access list entry, you must specify AL=WORKUNIT.

,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*. When you specify ALET 0, the system returns the caller's primary address space STOKEN. Do not specify ALET 1 on an EXTRACT request.

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 an 8-byte identifier of an address space, data space, subspace, or hiperspace. For ADD processing, 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.

,CHKEAX=YES
,CHKEAX=NO
Specifies that ALESERV does (CHKEAX=YES) or does not (CHKEAX=NO) check the EAX authority of the caller to the address space to be added to or deleted from the access list. The default is CHKEAX=YES.

,CHKPT=FAIL
,CHKPT=IGNORE
Specifies how the system is to process a checkpoint request made through the CHKPT macro, relevant 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.

Note: This keyword does not apply to an access list entry being added for a SCOPE=COMMON data space. Access list entries for SCOPE=COMMON data spaces are always ignored by the system on a checkpoint request.

If you specify CHKPT=IGNORE, you assume full responsibility of managing the data space, subspace, or hiperspace storage. For more information, *z/OS MVS Programming: Extended Addressability Guide*.

,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 which is specified are at the discretion of the user, and can be any valid coding values.

,REFTYPE=ANY
,REFTYPE=FETCHONLY
Identifies the reference type for the space for which the access list entry is being created. The default is ANY.

- ANY indicates that the space may be referenced either for fetch (read) or store (write).
- FETCHONLY indicates that the space may be referenced only for fetch (read). Store (write) is not permitted. The access list entry will have the FetchOnly attribute described in *z/Architecture® Principles of Operation*, SA22-7832. Do not use this option for a common area data space (CADS).

ABEND codes

None.

Return codes

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

Table 6. Return Codes for the ALESERV ADD Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: ALESERV ADD complete successfully. **Action**: None.

Table 6. Return Codes for the ALESERV ADD Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
08	**Meaning**: Program error. The caller is not EAX-authorized to the specified space; the entry is not added. The ALET returned is incorrect. **Action**: Verify that the intended STOKEN is specified.
0C	**Meaning**: Environmental error. The current access list cannot be expanded. There are no free access list entries and the maximum size is reached. **Action**: Delete unused entries and reissue the request.
10	**Meaning**: Environmental error. ALESERV might not obtain storage for an expanded access list. **Action**: Retry the request.
18	**Meaning**: Program error. A problem-state caller with PSW key 8 - F tried to add an entry to the PASN-AL for a space other than a SCOPE=SINGLE data space, or tried to add an entry to any access list for a hiper space or for a data 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 inactive. **Action**: Enable your program before it issues ALESERV.
24	**Meaning**: Program error. AR 1 contained an ALET of 1 on input or contained an ALET for the PASN-AL. **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 which is represented by the input STOKEN is not valid for cross memory access. **Action**: None required. However, you might want to take action based on 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 the execute form of the macro correctly addresses the parameter list.
54	**Meaning**: Program error. The caller tried to add a data space, hiperspace, or subspace to an access list as a private entry. **Action**: Specify ACCESS=PUBLIC instead of ACCESS=PRIVATE.
5C	**Meaning**: Program error. The caller was not authorized to add a data space or a hiperspace to an access list. **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.
62	**Meaning:** Program error. A previous error in your program left the access list in an unexpected format. The error occurs 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.

Table 6. Return Codes for the ALESERV ADD Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
68	**Meaning**: Program error. The caller attempted to add a hiperspace under conditions, which are not allowed. See Table 5 on page 31 for a summary of the rules for adding hiperspaces to an access list. **Action**: Verify that the options which are specified on your ADD request do not violate the rules specified in Table 5 on page 31.
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. **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. **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 caller is not the owner or 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.
8C	**Meaning**: Program error. The caller requested REFTYPE=FETCHONLY when attempting to add an access list entry for a common area data space (CADS). **Action**: Avoid requesting REFTYPE=FETCHONLY for a CADS.

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

Table 7. Return Codes for the ALESERV ADDPASN Macro	
Hexadecimal Return Code	**Meaning and Action**
00	**Meaning**: ALESERV ADDPASN completes successfully. **Action**: None.
0C	**Meaning**: Environmental error. The current access list cannot be expanded. There are no free ALEs and the maximum size is reached. **Action**: Delete unused entries and reissue the request.
10	**Meaning**: Environmental error. ALESERV might 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**: Program error. AR 1 contained an ALET of 1 on input or contained an ALET for a PASN-AL. **Action**: Verify that AR 1 contains either an ALET of 0 or the ALET for the caller's DU-AL.

Table 7. Return Codes for the ALESERV ADDPASN Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
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 occurs because the SRB environment was not valid when the system dispatched an SRB. The system did not perform the ALESERV ADDPASN request. **Action**: Determine the cause of the error that preceded the ALESERV ADD request. Correct the error and rerun the program.

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

Table 8. Return Codes for the ALESERV DELETE Macro	
Hexadecimal Return Code	**Meaning and Action**
00	**Meaning**: ALESERV DELETE completes successfully. **Action**: None.
08	**Meaning**: Program error. The caller is not EAX-authorized to the address space specified by the ALET. The entry is not deleted. **Action**: Verify that the intended STOKEN is specified.
14	**Meaning**: Program or environmental error. The input ALET corresponds to an access list entry 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 for 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 attempted to delete an entry from the PASN-AL for a space other than a SCOPE=SINGLE data space. **Action**: Verify that the ALET supplied represents the intended space.
60	**Meaning**: System error. An unexpected error occurred. The request was not completed. **Action**: Retry the request.
64	**Meaning**: Program error. A problem-state caller with PSW key 8 - F tried to delete an entry with CHKEAX=NO. **Action**: Specify CHKEAX=YES.

Table 8. Return Codes for the ALESERV DELETE Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
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 not the creator or the owner 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.

Table 9. Return Codes for the ALESERV EXTRACT Macro	
Hexadecimal Return Code	**Meaning and Action**
00	**Meaning**: ALESERV EXTRACT completes 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.
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 contains an ALET for 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 which is associated with the input ALET is not valid for cross memory access. **Action**: None required. However, you might want to take action based on 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 might want to discard the specified ALET and possibly take action based on 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 and environmental error. The ALET the caller which is specified represents an STOKEN for a data space that is no longer accessible. **Action**: None required. However, you might want to discard the specified ALET and possibly take action based on 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.

Table 10. Return Codes for the ALESERV SEARCH Macro	
Hexadecimal Return Code	**Meaning and Action**
00	**Meaning**: ALESERV SEARCH completes 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 for 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 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 and the ALET indexes into the DU-AL. **Action**: Change the AL or the ALET parameters to specify the correct AL and ALET combination.
60	**Meaning**: System error. An unexpected error occurred. The request was not completed. **Action**: Retry the request.

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

Table 11. Return Codes for the ALESERV EXTRACTH Macro	
Hexadecimal Return Code	**Meaning and Action**
00	**Meaning**: ALESERV EXTRACTH completes successfully. **Action**: None.
24	**Meaning**: Program error. AR 1 contained an ALET of 1 on input or contains 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

Add an entry to a DU-AL for a data space by issuing 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 the ALESERV parameter list.

Syntax

The list form 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.
MF=L	
,RELATED=*any-value*	

Parameters

The parameters are explained as follows:

MF=L
Specifies the list form of ALESERV.

,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 macro parameter expression.

ALESERV - Execute form

The execute form of ALESERV uses a remote parameter list that can be generated by the list form of ALESERV.

Syntax

The execute 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.

Syntax	Description
	Valid parameters (Required parameters are underlined)
ADD	AL, STOKEN, ACCESS, ALET, CHKEAX, CHKPT, MF,
	RELATED
ADDPASN	ALET, CHKPT, MF, RELATED
DELETE	ALET, MF, CHKEAX, RELATED
EXTRACT	ALET, STOKEN, MF, RELATED
SEARCH	AL, ALET, STOKEN, 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.
,CHKEAX=YES	**Default:** CHKEAX=YES.
,CHKEAX=NO	
,CHKPT=FAIL	**Default:** CHKPT=FAIL
,CHKPT=IGNORE	
,RELATED=*any-value*	*any-value*: Any valid macro parameter specification.
,MF=(E,*list-addr)*	*list-addr*: RX-type address or register (2)-(12).

Parameters

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

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

Specifies the execute form, which uses a remote parameter list. *list addr* specifies the address of the remote parameter list, generated by the list form of the macro.

Chapter 3. ASCRE — Create address spaces

Description

The ASCRE macro creates an address space. The caller of ASCRE can establish cross memory linkages between the creating address space and the created address space. In this macro description, the created address space is called the “new” address space.

Use the INIT parameter to specify an address space initialization routine that runs in the new address space. The initialization routine performs such actions as loading modules into the new address space and building control blocks there.

Use either the ASNAME or STPARM parameter to name the new address space and specify the first program that will run after the initialization routine completes. This first program has all system services available to it.

Optionally, you can use the AXLIST, TKLIST, and LXLIST parameters to set up cross memory linkages that allow programs in the new address space to use the services of programs in the creator's address space.

- AXLIST specifies the location of a list of authorization index (AX) values that the caller obtained through AXRES.
- TKLIST specifies the location of the list of tokens that represents the entry tables built by the creating address space.
- LXLIST specifies the location of a list of linkage index (LX) values that the caller obtained through LXRES.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	24 or 31
ASC mode:	Primary or AR
Interrupt Status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	For callers in primary mode, control parameters must be in the primary address space. For callers in AR address space control (ASC) mode, the parameters can be in the primary address space (qualified by an ALET of 0) or in any space addressable through public entries in the caller's dispatchable unit access list (DU-AL).

Programming requirements

The caller in AR ASC mode must have issued SYSSTATE ASCENV=AR to tell ASCRE to generate code and addresses appropriate for callers in AR mode.

Restrictions

The caller must not have an enabled unlocked task (EUT) functional recovery routine (FRR) established.

Register information

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

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

Register
Contents

0
Reason code

1
If the return code is 4, GPR 1 contains the address of the ASCB for the new address space. Otherwise, GPR 1 is used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Register
Contents

0
Used as a work register by the system

1
Contains a 0 if the return code is 4; otherwise, used as a work register by the system.

2-13
Unchanged

14-15
Used as work registers by the system

See *z/OS MVS Programming: Extended Addressability Guide* for information on initializing address spaces. It also gives an example of coding the ASCRE macro.

Performance implications

None.

Other implications

A task started under JES2 using the default IEESYSAS proc will have a jobname of IEESYSAS in the JES2 $DS(*sss*), where *sss* is the started task number. The SDSF panel DA will also show a jobname of IEESYSAS. The stepname, however, will be that of the started task. The z/OS command- D A,L will show both a jobname and stepname of the started task.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ASCRE.
ASCRE	
ƀ	One or more blanks must follow ASCRE.
ASNAME=*as name*	*as name*: One to eight characters, enclosed in apostrophes.
STPARM=*start parm addr*	*start parm addr*: RX-type address or register (2) - (12).
,INIT=*init rtn name*	*init rtn name*: One to eight characters, enclosed in apostrophes.
,INIT=*init rtn name addr*	*init rtn name addr*: RX-type address or register (2) - (12).
,ODA=*output data addr*	*output data addr*: RX-type address or register (2) - (12).
,TRMEXIT=*term rtn addr*	*term rtn addr*: RX-type address or register (2) - (12).
,UTOKEN=*user token addr*	*user token addr*: RX-type address or register (2) - (12).
	Note: Specify UTOKEN only if you specify TRMEXIT.
,ASPARM=*parm area addr*	*parm area addr*: RX-type address or register (2) - (12).
,ATTR=*attribute list*	*attribute list*: List of attributes, separated by commas.
,AXLIST=*ax list addr*	*ax list addr*: RX-type address or register (2) - (12).
,TKLIST=*token list addr*	*token list addr*: RX-type address or register (2) - (12).
	Note: When you specify TKLIST, specify LXLIST also.
,LXLIST=*lx list addr*	*lx list addr*: RX-type address or register (2) - (12).
,ELXLIST=*lx list addr*	*elx list addr*: RX-type address or register (2) - (12).
	Note: Specify LXLIST only if you specify TKLIST.

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

Parameters

The parameters are explained as follows:

ASNAME=*as name*
STPARM=*start parm addr*
Specifies either the name of the new address space or the address of a parameter string that is input to an internal START command. You must specify either STPARM or ASNAME. Use ASNAME if you are adding a procedure to SYS1.PROCLIB and you are not passing parameters to JCL.

ASNAME specifies the address space name (the same as the name of the procedure in SYS1.PROCLIB that specifies the first program to execute in the new address space.) The operator uses this name to issue certain commands, such as the DISPLAY command that displays information about the address space. The name must consist of 1 to 8 characters, enclosed by apostrophes. The first character must be alphabetic or national (#, $, or @); other characters can be alphabetic, national, or numeric.

STPARM specifies the address of a parameter string that is input to an internal START command that the system uses to start the address space. The string consists of a two-byte length field, followed by up to 124 bytes of parameter data. The length field identifies the length of the parameter data (not including the length field itself). The parameter data consists of START command parameters, for example “GTF,,,JES2”. Data must begin with the address space name, which corresponds to the procedure in SYS1.PROCLIB that specifies the first program that is to execute in the new address space.

If you do not need special DD definitions for data sets, specify the common system address space procedure IEESYSAS. In the parameter data, specify the system-defined procedure IEESYSAS in the following format:

```
IEESYSAS.x,PROG=y
```

where:

- *x* is name of the address space.
- *y* is the name of the first program to execute in the new address space.

,INIT=*init rtn name*
,INIT=*init rtn name addr*
Specifies the name of the address space initialization routine or the address of the name. The name is a string of up to eight alphanumeric characters; if you specify *init rtn name*, you must enclose the name in apostrophes. The first character of the name must be alphabetic or national; other characters can be alphabetic, national, or numeric. If the name is less than eight characters, left-justify the name and pad with blanks on the right to make up the eight characters.

The routine, which performs functions such as loading modules, must reside in either the LPA (PLPA, MLPA, fixed LPA) or in a library in the LNKLST concatenation. If the routine uses the two ECBs (EAERIMWT and EAEASWT) that the system provides for communication between the creating address space and the initialization routine, the routine must be in 31-bit addressing mode.

INIT is a required parameter. If you do not need an initialization routine, you can specify the dummy module IEFBR14 on the INIT parameter.

,ODA=*output data addr*
Specifies the address of a 24-byte area that contains output information from the ASCRE macro. The output information, mapped by the macro IHAASEO, consists of:

- Eight bytes for the STOKEN of the new address space

If you use the ASDES macro to terminate the new address space, you can obtain the STOKEN from this field.

- Four bytes for the address of the ASCB of the new address space
- Four bytes for the address of the two contiguous ECBs (EAERIMWT and EAEASWT).

  The creator of the address space and the new address space can use these two ECBs for communicating and synchronizing. They are mapped by IEZEAECB. A program must be in 31-bit addressing mode when it references them.
- Eight bytes (not part of the programming interface)

ODA is required.

,TRMEXIT=*term rtn addr*
Specifies the address of the termination routine that gets control when the new address space terminates. The routine receives control in 31-bit addressing mode, supervisor state, and in the current TCB key of the task that issued the ASCRE macro. The routine is an asynchronous exit (IRB) that resides in the address space of the creating program and runs under its TCB.

On entry to the routine:

- GPR 1 contains the address of a copy of the 64-bit token that the UTOKEN parameter supplies.
- GPR 13 conditionally contains the address of a standard 18-word save area, providing either of the following conditions are met:
  - The TCB key of the caller of the ASCRE macro is 0-7.
  - The job step of the caller is APF-authorized and is not running the TSO/E TMP.
- GPR 14 contains the return address.
- GPR 15 contains the entry point address.

If you specify TRMEXIT, you can also specify UTOKEN.

,UTOKEN=*user token addr*
Specifies the address of a 64-bit token of your choice that the termination routine can use to identify the new address space. Do not specify UTOKEN unless you specify TRMEXIT. If you specify TRMEXIT without specifying UTOKEN, the termination routine does not have the user data.

,ASPARM=*parm area addr*
Specifies the address of a parameter string that the new address space can obtain through the ASEXT macro. The parameter string consists of a halfword length field, followed by up to 254 bytes of parameter data. The length field contains the length of the parameter data (not including the length field itself).

,ATTR=*attr*
Specifies some attributes of the new address space. Attributes specified on the execute form of the ASCRE macro are added to the options specified on the list form.

Options for the ATTR parameter are as follows:

JOBSPACE
The address space is to be marked as a "job" (started task) address space, instead of as a "system" address space.

NONURG
Specifies that the address space will be used by non-urgent services. Specify either NONURG or HIPRI. NONURG is the default.

HIPRI
Indicates that the address space is for a high-priority service. Specify either NONURG or HIPRI. NONURG is the default.

PERM
Specifies that the system does not terminate the new address space when the TCB that represents the creating program terminates. If you do not specify PERM, the system terminates the new address space when it terminates the TCB.

NOMT
The address space may not be MEMTERMed unless a DAT error occurs. If a DAT error does occur then the recovery action is controlled by the NOMD option. If an unrecoverable error occurs for an address space created with NOMT the entire system is placed into a wait state.

This specification does not prevent the ASDES service from forcing the termination of the address space.

NOMD
The address space may not be MEMTERMed on a DAT error.

This option is honored only if NOMT is also specified. If a DAT error occurs for an address space created with NOMD the entire system is placed into a wait state.

1LPU
The address space must have all private area long-term fixed pages assigned to preferred (nonreconfigurable and non-V=R) storage frames.

This option is the same as specifying LPREF for a program on a PPT definition.

2LPU
The address space must have all private area short-term fixed pages assigned to preferred (nonreconfigurable and non-V=R) storage frames.

This option is the same as specifying SPREF for a program on a PPT definition.

N2LP
The address space does not need to have all private area short-term fixed pages assigned to preferred storage frames. That is, the program's short-term fixes are in fact short-term fixes and can be allowed in reconfigurable storage.

This option is the same as specifying NOPREF for a program on a PPT definition.

PRIV
The address space is privileged.

A task marked PRIV is put in the SYSSTC service class if it is not explicitly classified in the WLM classification rules.

NOSWAP
The address space is non-swappable.

CANCEL
The address space jobstep can be canceled after the ASCRE initialization routine is completed.

REUSASID
The address space is assigned to a reusable ASID, if REUSASID(YES) was specified in parmlib member DIAGxx. For more information about reusing ASIDs, see *z/OS MVS Programming: Extended Addressability Guide*.

,AXLIST=*ax list addr*
Specifies the address of a list of halfwords containing the AX values for the new address space. These values determine the PT and SSAR authority for programs. (This list was obtained through the AXRES macro.) The first entry in the list describes the number of AX values in the list (from 1 to 32).

Using this parameter has the same effect as a program in the new address space issuing the ATSET macro once for each AX value in the list.

,TKLIST=*token list addr*
Specifies the address of a list of fullword tokens that represent the entry tables that the system is to connect to the linkage table of the new address space. The first entry in the list describes the number of token values that follow (from 1 to 32). The ETCRE macro returned these tokens in register 0. Using this parameter has the same effect as a program in the new address space issuing the TKLIST parameter on the ETCON macro.

When you specify TKLIST, you must also specify LXLIST.

,LXLIST=*lx list addr*
,ELXLIST=*elx list addr*

lx list addr specifies the address of a list of values that represent indexes into the linkage table. Each linkage index (LX) value represents an entry in the linkage table. The system connects the entry tables specified by the TKLIST parameter to the LX values specified in this list. The first entry in the list must be the number of LX values that follow (from 1 to 32). The number of LX values must be the same as the number of entry table tokens. Using this parameter has the same effect as a program in the new address space issuing the LXLIST parameter on the ETCON macro.

elx list addr specifies the address of an area that contains extended linkage index (LX) values returned by the ELXLIST parameter of LXRES. The first word in the area must be the number of extended LX values that follow (from 1 to 32). Each subsequent eight bytes contains an extended LX value, which consists of a 4-byte sequence number followed by an LX value. Each extended linkage index value represents an entry in the linkage table. The system connects the entry tables specified by the TKLIST parameter to the extended LX values specified in this list. The number of extended LX values must be the same as the number of entry table tokens. Using this parameter has the same effect as a program in the new address space issuing the ELXLIST parameter on the ETCON macro.

When you specify TKLIST, you must also specify either LXLIST or ELXLIST.

,RELATED=*value*

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

Return and reason codes

The following table gives the return codes from register 15 and the associated reason codes from register 0.

Table 12. Return and Reason Codes for the ASCRE Macro

Decimal Return Code	Decimal Reason Code	Meaning
00	00	**Meaning:** The address space has been created. Data has been returned in the output data area.
00	04	**Meaning:** The address space creation has been scheduled. Data has been returned in the output data area.
04	04	**Meaning:** The address space has been created; there was an error accessing the ODA.
04	08	**Meaning:** The address space creation has been scheduled; there was an error accessing ODA.
08	04	**Meaning:** The caller is not in supervisor state.
08	08	**Meaning:** The caller is not enabled.
08	12	**Meaning:** The caller is not in task mode.
08	16	**Meaning:** The caller is not unlocked.
08	20	**Meaning:** GPR 0 has an invalid function code on input.
08	24	**Meaning:** ASCRE could not establish recovery.
12	04	ASCRE cannot reference the parameter list.
12	08	**Meaning:** The version number in the parameter list is not valid.
12	12	**Meaning:** The reserved field in the parameter list is not 0.
16	04	**Meaning:** ASCRE cannot reference the INIT parameter.
16	08	**Meaning:** The initialization routine is not specified or is specified incorrectly.
20	04	**Meaning:** ASCRE cannot reference the STPARM or ASNAME parameter.
20	08	**Meaning:** Neither STPARM or ASNAME was specified.

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

Decimal Return Code	Decimal Reason Code	Meaning
20	12	**Meaning:** The STPARM length is not 1-124.
24	04	**Meaning:** The reserved attribute bit is set.
24	08	**Meaning:** Both HIPRI and NONURG are specified.
28	04	**Meaning:** ASCRE cannot reference the UTOKEN.
28	08	**Meaning:** UTOKEN is specified without TRMEXIT.
32	04	**Meaning:** ASCRE cannot reference the ASPARM parameter.
32	08	**Meaning:** The ASPARM length is not 0-254.
36	04	**Meaning:** ASCRE cannot reference AXLIST.
36	08	**Meaning:** The AXLIST length is not 1-32 elements.
40	04	**Meaning:** ASCRE cannot reference LXLIST.
40	08	**Meaning:** The LXLIST length is not 1-32 elements.
44	04	**Meaning:** ASCRE cannot reference the TKLIST parameter.
44	08	**Meaning:** The TKLIST length is not same as LXLIST length.
48	08	**Meaning:** The ASCRE invocation specified one of the following: • ASNAME, but the address space name is not valid. • STPARM for a procedure other than IEESYSAS, but the address space name is not valid. • STPARM for procedure IEESYSAS, but did not correctly provide the required format of IEESYSAS.*x*, where *x* is the name of the address space.
52	04	**Meaning:** A storage shortage prevented the creation of an address space. Resubmit the failed job because the shortage might have been caused by a temporary strain on workload. If the problem persists, you might have to reevaluate your installation defined storage thresholds.
52	08	**Meaning:** Either the maximum number of address spaces was exceeded or the system could not obtain storage for the ASCB or ASSB. The system programmer can change the value specified on the MAXUSER parameter in the IEASYSxx parmlib member (to increase the number of address spaces that are available).
52	12, 16	**Meaning:** Record the return and reason codes and inform your technical support personnel.
56	16	**Meaning:** The caller specified an address space attribute that is not valid.
60, 64, 68, 72	Any	**Meaning:** Record the return and reason codes and inform your technical support personnel.

Example

Create an address space named ASPACE1. Note the USING statements that establish addressability for different segments of code.

```
ASCRETST CSECT
ASCRETST AMODE 31
ASCRETST RMODE ANY
         BALR  10,0              ESTABLISH ...
         USING *,10              ... ADDRESSABILITY
         .
         .
*  ISSUE ASCRE SPECIFYING A TERMINATION ROUTINE NAME IN STORAGE
         ASCRE ASNAME='ADSPACE1',INIT=INITNAME,TRMEXIT=TERMEXIT,ODA=ODAAREA
         .
         .
*        TERMINATION EXIT
TERMEXIT DS  0H
         USING  *,15             REGISTER 15 CONTAINS ENTRY ADDRESS
         SAVE   (14,12),,*       SAVE REGISTERS
```

```
         .
*        PERFORM ADDRESS SPACE TERMINATION PROCESSING
         .
         .
         RETURN (14,12)           RESTORE REGISTERS; RETURN TO SYSTEM
         .
         .
*        DATA AREAS
INITNAME DC    CL8'INITMOD'
ODAAREA  DS    CL24
         END
```

ASCRE - List form

The list form of the ASCRE macro constructs a nonexecutable parameter list. This list, or a copy of it for reentrant programs, can be referred to by the execute form of the macro.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ASCRE.
ASCRE	
ƀ	One or more blanks must follow ASCRE.
ASNAME=*as name*	*as name*: One to eight characters, enclosed in apostrophes.
STPARM=*start parm addr*	*start parm addr*: RX-type address.
,INIT=*init rtn name*	*init rtn name*: One to eight characters, enclosed in apostrophes.
,INIT=*init rtn name addr*	*init rtn name addr*: RX-type address or register (2) - (12).
,ODA=*output data addr*	*output data addr*: A-type address.
,TRMEXIT=*term rtn addr*	*term rtn addr*: A-type address.
,UTOKEN=*user token addr*	*user token addr*: A-type address.
	Note: Specify UTOKEN only if you specify TRMEXIT.
,ASPARM=*parm area addr*	*parm area addr*: A-type address.

Syntax	Description
,ATTR=*attribute list*	*attribute list*: List of attributes, separated by commas.
,AXLIST=*ax list addr*	*ax list addr*: A-type address.
,TKLIST=*token list addr*	*token list addr*: A-type address.
	Note: When you specify TKLIST, specify LXLIST also.
,LXLIST=*lx list addr*	*lx list addr*: A-type address.
	Note: Specify LXLIST only if you specify TKLIST.
,RELATED=*value*	*value*: Any valid macro parameter specification.
,MF=L	

Parameters

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

,MF=L
Specifies the list form of ASCRE.

ASCRE - Execute form

The execute form of the ASCRE macro can refer to and modify a remote parameter list built 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.
ƀ	One or more blanks must precede ASCRE.
ASCRE	
ƀ	One or more blanks must follow ASCRE.

Syntax	Description
ASNAME=*as name*	*as name*: One to eight characters, enclosed in apostrophes.
STPARM=*start parm addr*	*start parm addr*: RX-type address or register (2) - (12).
,INIT=*init rtn name*	*init rtn name*: One to eight characters, enclosed in apostrophes.
,INIT=*init rtn name addr*	*init rtn name addr*: RX-type address or register (2) - (12).
,ODA=*output data addr*	*output data addr*: RX-type address or register (2) - (12).
,TRMEXIT=*term rtn addr*	*term rtn addr*: RX-type address or register (2) - (12).
,UTOKEN=*user token addr*	*user token addr*: RX-type address or register (2) - (12).
	Note: Specify UTOKEN only if you specify TRMEXIT.
,ASPARM=*parm area addr*	*parm area addr*: RX-type address or register (2) - (12).
,ATTR=*attribute list*	*attribute list*: List of attributes, separated by commas.
,AXLIST=*ax list addr*	*ax list addr*: RX-type address or register (2) - (12).
,TKLIST=*token list addr*	*token list addr*: RX-type address or register (2) - (12).
	Note: When you specify TKLIST, specify LXLIST also.
,LXLIST=*lx list addr*	*lx list addr*: RX-type address or register (2) - (12).
	Note: Specify LXLIST only if you specify TKLIST.
,RELATED=*value*	*value*: Any valid macro parameter specification.
,MF=(E,*cntl addr)*	*cntl addr*: RX-type address or register (2) - (12)

Parameters

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

,MF=(E,*cntl addr*)
Specifies the execute form of the ASCRE macro. *cntl addr* is the address of the remote parameter list that the list form of the macro provided.

Chapter 4. ASDES — Terminate an address space

Description

The ASDES macro terminates an address space that was created through the ASCRE macro.

z/OS MVS Programming: Extended Addressability Guide describes how to create and terminate address spaces.

ASDES processing circumvents all task recovery and task resource manager processing. Its use should be restricted to a select group of routines that can determine that task recovery and task manager clean-up are either not warranted or will not successfully operate in the address space being terminated. An alternate way to terminate an address space is to use CALLRTM TYPE=ABTERM and specify the jobstep TCB.

Environment

Requirements for the caller of ASDES are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	24 or 31
ASC mode:	Primary or AR
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	For callers in primary mode, control parameters must be in the primary address space. For callers in AR mode, the parameters can be in any space addressable through public entries in the caller's dispatchable unit access list (DU-AL).

Programming requirements

Callers in access register (AR) mode must have issued SYSSTATE ASCENV=AR to tell ASDES to generate code and addresses appropriate for callers in AR mode.

The caller must not have an enabled unlocked task (EUT) functional recovery routine (FRR) established.

Output register information

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

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

Register
Contents

0
Reason code

1
Used as a work register by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

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

Register
Contents

0-1
Used as work registers by the macro

2-13
Unchanged

14-15
Used as work registers by the macro

Syntax

The syntax of the ASDES macro is as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ASDES.
ASDES	
ƀ	One or more blanks must follow ASDES.
STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or registers (2) - (12).
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

STOKEN=*stoken-addr*
Specifies the address of an eight-byte area that contains the STOKEN of the address space you want to terminate. The system returned the STOKEN in the 24-byte area requested by the ODA parameter on the ASCRE macro that created the address space. STOKEN is a required parameter.

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

Return and reason codes

Return codes and reason codes (in decimal form) are shown in the following table.

Table 13. Return and Reason Codes for the ASDES Macro

Decimal Return Code	Decimal Reason Code	Meaning
00	00	**Meaning:** Address space is terminated.
08	04	**Meaning:** Caller is not in supervisor state.
08	08	**Meaning:** Caller is not enabled.
08	12	**Meaning:** Caller is not in task mode.
08	16	**Meaning:** Caller is not unlocked.
08	20	**Meaning:** GPR 0 had invalid function code.
08	24	**Meaning:** ASDES could not establish recovery.
12	04	**Meaning:** ASDES could not reference the STOKEN parameter.
12	08	**Meaning:** STOKEN does not map to a valid address space. Address space might have already terminated.
16	04	**Meaning:** The address space was not created by ASCRE.

Chapter 5. ASEXT — Extract address space parameters

Description

The ASEXT macro returns to the caller the address of a copy of a parameter string that the creating program made available at the time it created the primary address space. Use this macro only if the primary address space was created through the ASCRE macro and you specified the ASPARM parameter on the ASCRE macro.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	24-bit or 31-bit. To reference the copy of the parameter string, the user must be in 31-bit addressing mode.
ASC mode:	Primary or AR
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

The caller must not have an enabled unlocked task (EUT) functional recovery routine (FRR) established.

Restrictions

None.

Register information

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

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

Register
Contents

0
Reason code, described in “Return and reason codes” on page 61.

1
Address of the extracted parameter string if the return code is 0; otherwise, contains a 0.

2-13
Unchanged

14
Used as a work register by the system

15
Return code, described in “Return and reason codes” on page 61

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

Register
Contents

0
Used as a work register by the system.

1
AR 1 contains a 0, which indicates that the parameter string copy is addressable in the primary address space.

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The syntax of the ASEXT macro is as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ASEXT.
ASEXT	
ƀ	One or more blanks must follow ASEXT.
ASPARM	
,RELATED=*value*	*value*: Any valid macro parameter specification.

Parameters

The parameters are explained as follows:

ASPARM
Requests the address of a copy of the parameter string (including the halfword length field) that the creator of the address space specified on the ASPARM parameter on the ASCRE macro. ASPARM is required.

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

Return and reason codes

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

Table 14. Return and Reason Codes for the ASEXT Macro

Decimal Return Code	Decimal Reason Code	Meaning
00	00	**Meaning**: The ASEXT service has completed successfully.
08	04	**Meaning**: The caller is not in supervisor state.
08	08	**Meaning**: The caller is not enabled.
08	12	**Meaning**: The caller is not in task mode.
08	16	**Meaning:** The caller is not unlocked.
08	20	**Meaning:** GPR 0 on input has an invalid function code.
08	24	**Meaning:** ASEXT is unable to establish recovery.
12	04	**Meaning:** GPR 1 has an invalid extract code on input.
16	04	**Meaning:** An unexpected error occurred while ASEXT was in progress.

Chapter 6. ATSET — Set authorization table

Description

The ATSET macro sets up an entry in the authorization table or in the authorization table bits. ATSET sets the PT and SSAR authority in the authorization table entry of the home address space. The authorization index value (AX) determines what entry is set.

The extended authorization index (EAX) determines what authorization table bits are set. To an address space, the EAX authority and SSAR authority are the same.

Related macros

AXEXT, AXFRE, AXRES, and AXSET

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN or PASN ¬=HASN
AMODE:	Any
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be addressable in the caller's primary address space

Programming requirements

None.

Restrictions

None.

Input register information

The ATSET 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 ATSET 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

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

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

Register
Contents

0-1
Used as work registers by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Performance implications

None.

Syntax

This is the standard form of the ATSET macro:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede ATSET.
ATSET	
ƀ	One or more blanks must follow ATSET.
AX=*ax value*	*ax value*: RX-type address or general register (0) - (12).
,PT=NO	**Default:** PT=NO
,PT=YES	
,SSAR=NO	**Default:** SSAR=NO

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

Parameters

These are the parameters:

AX=*ax value*
Specifies the AX value for which the PT and SSAR authority are to be set. The RX-type address points to the address of a half word containing the AX value. It is addressable in primary mode. When the register form is used, the AX value must be in bits 16-31. Bits 0-15 are ignored.

,PT=NO
,PT=YES
Specifies, YES or NO, whether program transfer (PT) is allowed into the home address space by routines executing with the specified AX.

,SSAR=NO
,SSAR=YES
Specifies, YES or NO, whether routines, executing with the specified AX, are allowed to establish secondary addressability to the home address space. It also specifies, YES or NO, whether routines with the specified EAX are allowed to access the address space through access registers.

,RELATED=*value*
Specifies information used to self-document macros. It “relates” functions or services to corresponding functions or services. The user can use any valid coding value. The format and contents are at the user's discretion.

Note: Every time you invoke the ATSET macro, you must set PT and SSAR authority. Specify: PT=YES.

ABEND codes

- 052
- 053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

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

Table 15. Return Code for the ATSET Macro.

Hexadecimal Return Code	Meaning
00	The selected authorization table entry has been set.

Examples

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

Chapter 7. ATTACH and ATTACHX — Create a subtask

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

ATTACH and ATTACHX description

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.

The descriptions of ATTACH and ATTACHX 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 active task when the ATTACH macro is issued. 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 load module containing the program to be given control is brought into virtual storage unless a usable copy is available in virtual storage. The issuing program can provide: an event control block, in which termination of the subtask is posted; an exit routine to be given control, when the subtask is terminated; and a parameter list the address of which is passed in GPR 1 to the subtask. The subtask is automatically removed from the system upon completion of its execution, unless the ECB or ETXR parameters are coded.

ATTACH and ATTACHX are also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the following parameters, which are restricted to use by authorized programs: SM, SVAREA, KEY, DISP, TID, NSHSPV, NSHSPL, JSTCB, and RSAPF.

Environment

The requirements for the caller of ATTACH or ATTACHX are:

Environmental factor	Requirement
Minimum authorization:	Problem state, and any PSW key. To use the SM, SVAREA, KEY, DISP, TID, NSHSPV, NSHSPL, JSTCB, or RSAPF parameter, the caller must either run in supervisor state or with PSW key 0-7. When the caller specifies JSTCB=YES and the program comes from an APF-authorized library or the link pack area and is link-edited with the APF-authorization attribute, the task runs with APF authorization.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
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.

Environmental factor	Requirement
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 ECB must be in storage addressable by both the caller of ATTACH and the system.
- Only job step tasks can issue ATTACH with JSTCB=YES. A task cannot issue a series of ATTACH macros that would cause its subtasks to be a mix of job step and nonjob step tasks.
- 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

Before issuing the ATTACH or ATTACHX macro, if you want to pass a parameter list to the new task without coding the PARAM or MF=E parameter, the caller must ensure that the following GPR contains the specified information:

Register
Contents

1
Address of the parameter list

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 subtask

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 system returns.

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 | NO	**Default:** SZERO=YES
,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 | NONE	**Note:** PURGE may be specified only if STAI or ESTAI is specified. **Default for STAI:** PURGE=QUIESCE **Default for ESTAI:** PURGE=NONE
,PURGE=HALT	
,ASYNCH=NO | YES	**Note:** ASYNCH may be coded only if STAI or ESTAI is specified. **Default for STAI:** ASYNCH=NO **Default for ESTAI:** ASYNCH=YES
,TERM=NO | YES	**Note:** TERM may be specified only if ESTAI is specified. **Default:** TERM=NO
,JSTCB=NO | YES	**Default:** JSTCB=NO

Syntax	Description
,SM=PROB | SUPV	**Default:** SM=PROB
,SVAREA=YES | NO	**Default:** SVAREA=YES
,KEY=PROP | ZERO	**Default:** KEY=PROP
,DISP=YES | NO	**Default:** DISP=YES
,DISP=RESET,TCB=*tcb addr*	*tcb addr*: RX-type address or address in register (2) - (12).
,TID=*task id*	*task id*: Decimal digits 0-255, or register (2) - (12).
	Default: TID=0
	Note: IBM recommends that you specify a value less than 200.
,NSHSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,NSHSPL=*subpool list addr*	*subpool list addr*: A-type address, or register (2) - (12).
,RSAPF=NO | YES	**Default:** RSAPF=NO
,ALCOPY=NO | YES	**Default**: ALCOPY=NO
,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 entry name list. The entry name is constructed using the BLDL macro. When EPLOC is coded, *entry name addr* points to an eight-byte field. When the name is less than eight characters, left-justify the name and pad with blanks on the right to make up the eight characters.

Note:

1. ATTACH processing can attach a load module in 24-bit or 31-bit addressing mode physically resident above or below 16 megabytes virtual. The AMODE and RMODE, load module attributes located in the directory entry for the load module, provide this information. The RMODE indicates the place of the module; the AMODE indicates the addressing mode of the module. When the AMODE of the entry point is ANY, it is attached with the same addressing mode as the caller.
2. When you use the DE parameter with the ATTACH macro, DE specifies the address of a list 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.**

3. See *z/OS DFSMS Macro Instructions for Data Sets* and *z/OS DFSMS Using Data Sets* for a description of the BLDL macro.

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

Register
Contents

0
Does not contain any information for use by the routine.

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
Do not contain any information for use by the routine.

13
Address of a 144-byte save area if SVAREA=YES was specified; Otherwise, zero.

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
Does not contain any information for use by the routine.

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
Do not contain any information for use by the routine.

13-15
Zeroes

,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 executed. The DCB must reside in storage below 16 megabytes.

,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 subtask, 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 subtask.

,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 subtask, with a higher number representing a higher dispatching priority. If, however, the resulting number is higher than the limit priority of the subtask, 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 subtask'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 for the subtask. The system uses this to indicate the termination of the subtask. This enables the issuer of the attach to wait on it, using the WAIT macro, and enables the system to post it on behalf of the terminating task. The return code, (when the task terminates normally), or the completion code, (when the task terminates abnormally), is placed in the event control block. When this parameter is coded, a DETACH macro must be issued 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. It is given control after the subtask normally or abnormally terminates. The exit routine is given control when the originating task becomes active after the subtask terminates. It must be in virtual storage. When this parameter is coded, a DETACH macro must be issued 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

Environmental factor	Requirement
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.

When the exit routine is given control, the contents of the GPRs are:

Register
Contents

0
Does not contain any information for use by the routine.

1
Address of the task control block for terminated task

2-12
Do not contain any information for use by the routine.

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

14
Return address

15
Address of the exit routine

When the exit routine receives control, the contents of ARs are:

Register
Contents

0
Does not contain any information for use by the routine.

1
Zero

2-12
Do not contain any information for use by the routine.

13-15
Zeroes

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

,GSPV=*subpool nmbr*
,GSPL=*subpool list addr*
Specifies a virtual storage subpool number, or address of a list of virtual storage subpool numbers, each less than 128. Ownership of each of the specified subpools is assigned to the subtask. Subpool zero is an exception. It can be specified but it cannot be transferred. When a task transfers ownership of a subpool, it can no longer obtain or release the associated virtual storage areas. When GSPL 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.

,SHSPV=*subpool nmbr*
,SHSPL=*subpool list addr*
Specifies a virtual storage subpool number or the address of a list of virtual storage subpool numbers, each less than 128. Programs of the originating task and the subtask can use the associated virtual storage areas. When 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 | NO
Specifies whether subpool 0 is to be shared with the subtask.

,SZERO=YES
Specifies that the subpool 0 is to be shared with the subtask.

,SZERO=NO
Specifies that the subpool 0 is not to be shared with the subtask.

Note: The default is SZERO=YES.

,TASKLIB=*dcb addr*
Specifies the address of the DCB for the library to be used as the attached subtask's library. Otherwise, the subtask 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 subtask with a completion code of X'806' if the requested module is not in the subtask 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 for the attached task; any STAI or ESTAI recovery routines defined for the attached task are automatically propagated to its subtasks.

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 | NONE | HALT
Specifies the action to be taken with regard to I/O operations when the subtask encounters an error.

,PURGE=QUIESCE
Indicates quiescing of I/O operations.

,PURGE=NONE
Indicates that no action is specified.

,PURGE=HALT
Indicates halting of I/O operations.

Note: The default for **STAI** is PURGE=QUIESCE. The default for **ESTAI** is PURGE=NONE.

,ASYNCH=NO | YES
Specifies whether asynchronous exits are to be allowed when a subtask encounters an error.

,ASYNCH=NO
Specifies that the asynchronous exits are not to be allowed when a subtask encounters an error.

,ASYNCH=YES
Specifies that the asynchronous exits are allowed when a subtask encounters an error. If this parameter is specified and the error is in asynchronous exit handling, recursion will develop when an asynchronous exit handling is the cause of the failure.

Note: This parameter is required 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: The default for **STAI** is ASYNCH=NO. The default for **ESTAI** is ASYNCH=YES.

,TERM=NO | YES
Specifies whether the recovery routine associated with the ESTAI request is scheduled in these situations:

- System-initiated logoff
- Job step timer expiration
- Wait time limit for job step exceeded
- DETACH macro without the STAE=YES parameter issued from a higher-level task (possibly by the system if the higher-level task encountered an error)
- Operator cancel
- Error on a higher-level task
- Error in the job step task when a nonjob step task issued the ABEND macro with the STEP parameter.
- z/OS UNIX is canceled and the user's task is in a wait in the z/OS UNIX kernel.

Note: The default is TERM=NO.

,JSTCB=NO | YES
Specifies whether the attached task is to be a job step task.

,JSTCB=NO
Specifies that the attached task is to be a nonjob step task and that the job step task of the issuer of ATTACH will be propagated to the newly attached task.

,JSTCB=YES
Specifies that the attached task is to be a job step task.

Note:

1. JSTCB=YES causes a new job pack area to be established for the attached task. Modules within the job pack area of the task issuing the ATTACH are not available to the newly attached task. See information about program management in *z/OS MVS Programming: Authorized Assembler Services Guide* for details.
2. The use of JSTCB=YES affects the ownership of those virtual storage subpools that are owned by job step tasks. See information about virtual storage management in *z/OS MVS Programming: Authorized Assembler Services Guide* for details.
3. Do not specify JSTCB=YES unless you know that the design of your application requires the special attributes of a job step task.

Note: The default is JSTCB=NO.

,SM=PROB | SUPV
Specifies which state the attached task is to run.

,SM=PROB
Specifies that the attached task is to run in problem state.

,SM=SUPV
Specifies that the attached subtask is to run in supervisor state.

Note: The default is SM=PROB.

,SVAREA=YES | NO
Specifies whether a save area is needed for the attached task.

,SVAREA=YES
Specifies that the ATTACH routine obtains a 144-byte save area. When the attaching and attached task share subpool zero, the save area is obtained there. Otherwise, it is obtained from a new 4KB block.

,SVAREA=NO
Specifies that no save area is needed.

Note: The default is SVAREA=YES.

,KEY=PROP | ZERO
Specifies whether the protection key of the newly created task should be propagated from the task or zero.

,KEY=PROP
Specifies that the protection key of the newly created task should be propagated from the task using ATTACH. If this parameter is specified, the key that the current task began is in the TCBPKF field or in the current PSW.

,KEY=ZERO
Specifies that the protection key of the newly created task should be zero.

Note: The default is KEY=PROP.

,DISP=YES | NO
,DISP=RESET,TCB=*tcb addr*
Specifies whether the attached subtask is dispatchable, non-dispatchable, or is reset to dispatchable.

,DISP=YES
Specifies that the attached subtask is dispatchable.

,DISP=NO
Specifies that the subtask is non-dispatchable; the system places the address of the TCB for the task in GPR 1, but ATTACH processing for the task does not complete. When you specify DISP=NO, you must issue ATTACH again with the DISP=RESET,TCB=*tcb addr* parameter so that ATTACH processing completes for the subtask.

,DISP=RESET,TCB=*tcb addr*
When you issue ATTACH with DISP=RESET,TCB=*tcb addr*, you cannot specify any other parameters on the ATTACH macro. ATTACH DISP=RESET,TCB=*tcb addr* resets to dispatchable the subtask specified by *tcb addr* and all subtasks of the attaching program that were attached using the DISP=NO parameter.

Note: The default is DISP=YES.

,TID=*task id*
Specifies the task identifier to be placed in the TCB field of the attached subtask. IBM recommends that you specify a value less than 200 for *task id*.

,NSHSPV=*subpool nmbr*
,NSHSPL=*subpool list addr*
Specifies the virtual storage subpool number 236 or 237, or the address of a list of virtual storage subpool numbers 236 and 237. The subpools specified are not shared with the subtask.

When NSHSPL is specified, the first byte of the list contains the number of bytes remaining in the list. Each of the subsequent bytes contains a virtual storage subpool number.

,RSAPF=NO | YES
Specifies whether the attached subtask comes from an unauthorized library or is to remain unchanged.

,RSAPF=NO
Specifies when the APF authorization of the step is to remain unchanged.

,RSAPF=YES
Specifies when the APF authorization of the step is to be changed. The attached subtask comes from an unauthorized library even though the calling program is running in supervisor state, system key (0-7), or both. The subtask is attached in the problem program state and with a non-system key.

Note: The default is RSAPF=NO.

,ALCOPY=NO | YES
Specifies the EAX value for the subtask and determines the contents of its access list.

,ALCOPY=NO
Gives the subtask an EAX of zero and a null access list.

,ALCOPY=YES
Gives the subtask:

- The same EAX as the caller
- A copy of the caller's DU-AL. For details about how the system copies a DU-AL, see the topic on access lists in *z/OS MVS Extended Addressability Guide*.

Note: The default is ALCOPY=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. They can be any valid coding values.

ABEND codes

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

ABEND Code	Associated Reason Code
12A	0,4
22A	0
42A	None
52A	0,4,8
72A	0,4,8,C,10,14, 18, 1C, 20, 24, 28, 2C
82A	None
92A	0,4,8,C,10,14,18 **Note:** ABEND code 92A results from an error not directly caused by the caller.

See *z/OS MVS System Codes* for explanations and responses for these codes.

Return codes

When control is returned, register 15 contains one of the return codes in the following table.

Table 16. Return codes for the ATTACH or ATTACHX macros

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 in a STAE exit.

Table 16. Return codes for the ATTACH or ATTACHX macros (continued)

Hexadecimal Return Code	Meaning and Action
08	**Meaning:** Environmental error. Insufficient storage available for control block for STAI/ESTAI request. Processing not completed. **Action:** Retry the request.
0C	**Meaning**: Program error. An incorrect exit routine address or incorrect parameter list address was specified with STAI parameter. Processing not completed. **Action:** Ensure that the exit routine and parameter list addresses are correct.
14	**Meaning**: Program error. An authorized task that specified JSTCB=YES is not a job step task. Processing not completed. **Action:** Either remove the JSTCB=YES option from this ATTACH macro or specify JSTCB=YES on the ATTACH macro for the current task.
18	**Meaning**: Program error. An attempt to create a new subtask would cause the current task to have a mix of job step and nonjob step subtasks. Processing not completed. **Action:** Change your program so that the ATTACH macros that it issues all specify the same value for JSTCB=.
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.
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

Attach program SYSPROGM, runs with protection key 0 and in supervisor mode. Subpool 0 is not to be shared, and the subtask is not to have a save area.

```
ATTACH EP=SYSPROGM,KEY=ZERO,SM=SUPV,SZERO=NO,SVAREA=NO
```

Example 2

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 3

Cause PROGRAM1 to be attached, share subpool 5, supply WORD1 so that the originating task can know when the subtask is complete, and establish EXIT1 as an ESTAI exit.

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

Example 4

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.

```
          ATTACH EP=PROGRAM1,SZERO=YES,ALCOPY=YES
```

ATTACHX - Create a subtask

The ATTACHX macro creates a subtask for callers in AR mode or primary mode. It indicates the entry point in the program to be given control when the subtask becomes active.

The format of the PARAM parameter list for callers in AR mode differs from the format for callers in primary mode.

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.

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 ATTACH or ATTACHX.
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, 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).

Syntax	Description
,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	
,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)

Syntax	Description
,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*)	
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	
,ASYNCH=NO	**Note:** Specify SYNCH only if you specify ESTAI.
,ASYNCH=YES	**Default for ESTAI:** ASYNCH=YES
,TERM=NO	**Note:** TERM may be specified only if ESTAI is specified.
,TERM=YES	**Default:** TERM=NO
,JSTCB=NO	**Default:** JSTCB=NO
,JSTCB=YES	
,SM=PROB	**Default:** SM=PROB
,SM=SUPV	
,SVAREA=YES	**Default:** SVAREA=YES
,SVAREA=NO	
,KEY=PROP	**Default:** KEY=PROP
,KEY=ZERO	
,KEY=NINE	
,PKM=SYSTEM_RULES	**Default:** PKM=SYSTEM_RULES
,PKM=REPLACE	
,DISP=YES	**Default:** DISP=YES
,DISP=NO	
,DISP=RESET,TCB=*tcb addr*	*tcb addr*: RX-type address or address in register (2) - (12)

Syntax	Description
,TID=*task id*	*task id*: Decimal digits 0-255, or register (2) - (12).
	Default: TID=0
	Note: IBM recommends that you specify a value less than 200.
,NSHSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit, or register (2) - (12).
,NSHSPL=*subpool list addr*	*subpool list addr*: A-type address, or register (2) - (12).
,RSAPF=NO	**Default:** RSAPF=NO
,RSAPF=YES	
,ALCOPY=YES	**Default**: ALCOPY=NO
,ALCOPY=NO	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,ADDRENV=SAME ,ADDRENV=SUBSP	**Default**: ADDRENV=SAME

Parameters

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

,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=ZERO
,KEY=NINE

ZERO specifies that the protection key of the newly created task should be zero. 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, 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 1

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
```

Example 2

Attach a nondispatchable subtask called SUBTASK1, then reset SUBTASK1 to be dispatchable.

```
          PRINT NOGEN
NODISP    CSECT
NODISP    AMODE ANY
NODISP    RMODE ANY
*********************************************************************
*  The following code performs these functions:                     *
*      1.  Creates a nondispatchable subtask by issuing the         *
*          ATTACHX macro with the DISP=NO parameter.                *
*      2.  Restarts the task by making it dispatchable with the     *
*          ATTACHX macro and the DISP=RESET parameter.              *
*                                                                   *
*********************************************************************
          SPACE 3
************************************
* Entry linkage                    *
************************************
          STM   R14,R12,12(R13)
          BALR  R12,0
          USING BEGN,R12
BEGN      DS    0H
          LA    R12,0(,R12)         CLEAN HI-BYTE OF ENTRY REGISTERS
          LA    R13,0(,R13)
          SPACE 3
          ST    R13,SAVE+4
          LA    R15,SAVE
          ST    R15,8(0,R13)
          LR    R13,R15
          EJECT
```

```
*********************************************************************
* Attach a subtask.   The subtask is in problem state and is        *
* nondispatchable.                                                  *
*********************************************************************
          SPACE 3
          ATTACHX EP=SUBTASK1,                                      X
                ECB=AMYECB,                                         X
                DISP=NO
          SPACE 3
          ST    R1,TCBADDR          SAVE SUBTASK TCB ADDRESS
          EJECT
          .                         PROCESSING CONTINUES
          .
          .
```

```
*******************************************************
*                                                     *
* Start the subtask by resetting it to be dispatchable.*
*                                                     *
*******************************************************
          SPACE 3
          L     2,TCBADDR         GET TARGET TCB ADDRESS
          SPACE 3
```

```
RESET    ATTACHX DISP=RESET,TCB=(2)
         SPACE 3
         EJECT
*****************************************************
* Wait until subtask completes, then detach subtask.*
*****************************************************
         SPACE 3
         WAIT  1,ECB=AMYECB
         SPACE 3
         DETACH TCBADDR           DETACH SUBTASK
         EJECT
************************************
* End of program                   *
************************************
         SPACE 3
FINI     DS    0H
         L     R13,SAVE+4
         DROP  R12
         LM    R14,R12,12(R13)
         XR    R15,R15
         BR    R14
         EJECT
*********************************
*   Define constants            *
*********************************
SAVE     DC    18F'0'
*
TCBADDR  DC    F'0'        ADDRESS OF SUBTASK TCB
AMYECB   DC    F'0'        END-OF-SUBTASK ECB
         SPACE 3
         EJECT
***********************************************
*   Register equates                          *
***********************************************
         SPACE 3
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

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 subtask can be provided using the list form of the CALL macro. This parameter list can be referred to in the execute form.

Syntax

The list form of 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.

Syntax	Description
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.
,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 STAI.
,ESTAI=*(exit addr,parm addr)*	

Syntax	Description
SDWALOC31=NO	**Note:** SDWALOC31 is valid only when ESTAI is specified AND when using ATTACHX.
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
,ASYNCH=NO	**Default for STAI:** ASYNCH=NO
,ASYNCH=YES	**Default for ESTAI:** ASYNCH=YES
	Note: ASYNCH can be specified only when STAI or ESTAI is specified.
,TERM=NO	**Note:** TERM can be specified only when ESTAI is specified.
,TERM=YES	**Default:** TERM=NO
,SM=PROB	**Default:** SM=PROB
,SM=SUPV	
,SVAREA=YES	**Default:** SVAREA=YES
,SVAREA=NO	
,KEY=PROP	**Default:** KEY=PROP
,KEY=ZERO	
,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.
,DISP=YES	**Default:** DISP=YES
,DISP=NO	
,DISP=RESET,TCB=*tcb addr*	*tcb addr*: RX-type address or address in register (2) - (12)
,TID=*task id*	*task id*: Decimal digits 0-255.
	Default: TID=0
	Note: IBM recommends that you specify a value less than 200.

Syntax	Description
,NSHSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit.
,NSHSPL=*subpool list addr*	*subpool list addr*: A-type address.
,RSAPF=NO	**Default:** RSAPF=NO
,RSAPF=YES	
,ALCOPY=YES	**Default:** ALCOPY=NO
,ALCOPY=NO	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,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 or ATTACHX macro.

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

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

Syntax	Description
,SVAREA=YES	**Default:** SVAREA=YES
,SVAREA=NO	
,KEY=PROP	**Default:** KEY=PROP
,KEY=ZERO	**Note:**KEY=NINE is valid only when using ATTACHX
,KEY=NINE	
,PKM=SYSTEM_RULES	**Default:** PKM=SYSTEM_RULES
,PKM=REPLACE	**Note:**PKM is valid only when using ATTACHX.
,DISP=YES	**Default:** DISP=YES
,DISP=NO	
,DISP=RESET,TCB=*tcb addr*	*tcb addr*: RX-type address or address in register (2) - (12)
,TID=*task id*	*task id*: Decimal digits 0-255, or address in register (2) - (12).
	Default: TID=0
	Note: IBM recommends that you specify a value less than 200.
,NSHSPV=*subpool nmbr*	*subpool nmbr*: Symbol, decimal digit, or address in register (2) - (12).
,NSHSPL=*subpool list addr*	*subpool list addr*: RX-type address, or address in register (2) - (12).
,RSAPF=NO	**Default:** RSAPF=NO
,RSAPF=YES	
,ALCOPY=YES	**Default:** ALCOPY=NO
,ALCOPY=NO	
,RELATED=*value*	*value*: Any valid macro keyword specification.
,ADDRENV=SAME ,ADDRENV=SUBSP	**Default**: ADDRENV=SAME
,MF=(E,*prob addr*)	*prob addr*: RX-type address, or address in register (1) or (2) - (12).
,SF=(E,*ctrl addr*)	*ctrl addr*: RX-type address, or address in register (2) - (12) or (15).
,MF=(E,*prob addr)*,SF=(E,*ctrl addr*)	

Syntax	Description

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 macro, with these exceptions:

,MF=(E,*prob addr*)
,SF=(E,*ctrl addr*)
,MF=(E,*prob addr*),SF=(E,*ctrl addr*)
Specifies the execute form of the ATTACH or ATTACHX macro. It uses a remote user parameter list, a remote control parameter list, or both.

Note:

1. When STAI is specified on the execute form, these fields are overlaid in the control parameter list: *exit addr, parm addr*, PURGE, and ASYNCH. When *parm addr* is not specified, zero is used. When PURGE or ASYNCH are not specified, defaults are used.
2. When ESTAI is specified on the execute form, these fields are overlaid in the control parameter list: *exit addr, parm addr*, PURGE, ASYNCH, and TERM. When *parm addr* is not specified, zero is used. When PURGE, ASYNCH, or TERM are not specified, defaults are used.
3. The STAI or ESTAI must be completely specified on either the list or execute form, but not on both forms.
4. When SZERO is not specified on the list or execute form, the default is SZERO=YES. When SZERO=NO is specified on either the list form or a previous execute form using the same SF=list, SZERO=YES is ignored for any subsequent execute forms of the macro. Once SZERO=NO is specified, it is in effect for all users of that list and cannot be overridden.
5. When RSAPF is not specified on the list or execute form, the default is RSAPF=NO. When RSAPF=YES is specified on either the list form or a previous execute form using the same SF=list, RSAPF=NO is ignored for any subsequent execute forms of the macro. Once RSAPF=YES is specified, it is in effect for all users of that list and cannot be overridden.
6. You cannot specify DISP=RESET,TCB=*tcb addr* on the List Form. However, you can build a list by using only the SF=L parameter, and use that list for the execute form that specifies DISP=RESET,TCB=*tcb addr*.

Chapter 8. AXEXT — Extract authorization index

Description

The AXEXT macro returns the authorization index value, AX, of the address space.

Related macros

ATSET, AXFRE, AXRES, and AXSET

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	Any
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

The AXEXT 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 AXEXT 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

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

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

Register
Contents

0
Bits 16-31 contain the extracted AX; bits 0-15 are set to zero

1
Used as a work register by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Performance implications

None.

Syntax

This is the standard form of the AXEXT macro:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede AXEXT.
AXEXT	
ƀ	One or more blanks must follow AXEXT.
ASID=*asid value*	*asid value*: RX-type address or register (0) - (12).
	Default: Current PASID.
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

These are the parameters:

ASID=*asid value*
Specifies the ASID of the address space from where the AX is to be extracted. When the RX-type address is used, it points to a halfword containing the ASID. When the register form is used, bits 16-31 contain the ASID and bits 0-15 are set to zero. When ASID is not specified, the current PASID is assumed.

,RELATED=*value*
Specifies information used to self-document macros by "relating" functions or services to corresponding functions or services. The format and content of the information are set at the discretion of the user. They can be any valid coding values.

ABEND codes

- 052
- 053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

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

Table 17. Return Codes for the AXEXT Macro

Hexadecimal Return Code	Meaning
00	The AX value of the specified address space was successfully obtained.

Examples

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

Chapter 9. AXFRE — Free authorization index

Description

The AXFRE macro returns one or more authorization index (AX) values to the system. The AX value can be used as an extended authorization index (EAX) value. The caller must ensure that the AXs to be returned are no longer being used by any address space as an AX or an EX; otherwise, the caller abnormally terminates. On completion of the AXFRE macro, all authorization of the freed AX values in authorization tables for the entire system are purged. The caller must be dispatched in the address space that owns the AX.

Related macros

AXEXT, ATSET, AXRES, and AXSET

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM = 0 to 7
Dispatchable unit mode:	SRB or task
Cross memory mode:	PASN=HASN
AMODE:	Any
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	For callers in primary mode, control parameters must be in the primary address space.

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

Programming requirements

When the macro is issued, the list of AX values passed to the AXFRE macro must be addressable in primary mode.

Restrictions

None.

Input register information

The AXFRE 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 AXFRE 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

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

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

Register
Contents

0-1
Used as work registers by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Performance implications

None.

Syntax

This is the standard form of the AXFRE macro:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede AXFRE.
AXFRE	
ƀ	One or more blanks must follow AXFRE.
AXLIST=*list addr*	*list addr*: RX-type address or register (0) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

These are the parameters:

AXLIST=*list addr*
Specifies the address of a variable length list of halfword entries that contain the AX values to be freed. The first halfword must contain the number of values in the list.

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

ABEND codes

- 052
- 053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

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

Table 18. Return Codes for the AXFRE Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: The specified authorization index or indexes are successfully freed. **Action**: None.
04	**Meaning**: The specified authorization index or indexes are not successfully freed. One or more of the indexes are unavailable for use. **Action**: None required. However, do not attempt to reuse these indexes.

Examples

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

Chapter 10. AXRES — Reserve authorization index

Description

The AXRES macro reserves one or more authorization index (AX) values for the caller's use. The AX values are owned by the current home address space.

The AXSET macro sets the AX of the home address space to the value (or values) that is reserved by the AXRES macro.

The caller can use the value returned by the system as an AX through the AXSET macro, or as an extended authorization index (EAX) through the ETDEF, ETCRE, and ETCON macros. The AX value associated with a program determines whether that program is permitted to issue the PT instruction with another address space as the target, and/or set another address space as its secondary address space through the SSAR instruction. The EAX value determines whether a program running with the EAX can access data in another address space through a private access list entry.

Related macros

ATSET, AXFRE, AXEXT, and AXSET

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM = 0 to 7
Dispatchable unit mode:	SRB or task
Cross memory mode:	PASN=HASN
AMODE:	Any
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	For callers in primary mode, control parameters must be in the primary address space.

Programming requirements

The parameter list passed to the AXRES macro must be addressable in primary mode when the macro expansion is executed.

Restrictions

None.

Input register information

The AXRES 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 AXRES 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

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

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

Register
Contents

0-1
Used as work registers by the macro.

2-13
Unchanged

14
Used as a work register by the macro.

15
Return code.

Performance implications

None.

Syntax

This is the standard form of the AXRES macro:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede AXRES.
AXRES	
ƀ	One or more blanks must follow AXRES.
AXLIST=*list addr*	*list addr*: RX-type address or register (0) - (12).

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

Parameters

The parameters are explained as follows:

AXLIST=*list addr*
Specifies the address of a variable length list, addressable in primary mode, of halfword entries in which requested AX values are to be returned. The first halfword must contain the number of values to be returned. Enough half-words must follow the first entry to contain the requested number of values. If the requested number of AX values is not available, the caller is abnormally terminated.

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

ABEND codes

- 052
- 053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

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

Table 19. Return Code for the AXRES Macro

Hexadecimal Return Code	Meaning
00	The AX value or values were successfully reserved.

Examples

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

Chapter 11. AXREXX - System REXX services

Description

AXREXX provides a macro interface for System REXX services.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	The caller must be authorized with any of the following attributes: • Supervisor state • PKM 0-7 • PSW key 0-7 • APF-authorized
Dispatchable unit mode:	Task mode
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space. However, control parameters for AR-mode callers must be in an address or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided REXX arguments supplied by the REXXARGS parameter have the same requirements and restrictions as the control parameters. The user-provided REXX variables in the REXXVARS parameter have the same requirements and restrictions as the control parameters. The user-provided data set name in the REXXINDSN parameter has the same requirements and restrictions as the control parameters. The user-provided data set name in the REXXOUTDSN parameter has the same requirements and restrictions as the control parameters. The user-provided information in the REXXDIAG parameter has the same requirements and restrictions as the control parameters. The user-provided information in the UTOKEN parameter has the same requirements and restrictions as the control parameters. The user-provided information in the REXXLIB parameter has the same requirements and restrictions as the control parameters.

Programming requirements

AXRZARG must be included in the invoking module.

Restrictions

The caller must not have any FRRs (Functional Recovery Routines) established.

Input register information

Before issuing the AXREXX macro, the caller does not have to place any information into any general purpose register (GPR) or access register (AR), unless using the input register in register notation for a particular parameter, or using the input register as a base register.

Output register information

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

Register
Contents

0
When the value in register 15 is not zero, the reason code from the service

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
The return code from the AXREXX Service

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 AXREXX macro is written as follows:

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

Syntax	Description
AXREXX	
	One or more blanks must follow AXREXX.
REQUEST=EXECUTE	
REQUEST=CANCEL	
REQUEST=GETREXXLIB	
,SECURITY=BYUTOKEN	**Default:** SECURITY=BYUTOKEN
,SECURITY=BYAXRUSER	
,REXXLIB=*xrexxlib*	*xrexxlib*: RS-type address or address in register (2) - (12)
,REXXLIB=NO_REXXLIB	**Default:** REXXLIB=NO_REXXLIB
,REXXLIBLEN=*xrexxliblen*	*rexxliblen*: RS-type address or address in register (2) - (12)
,REXXLIBLEN=NO_REXXLIBLEN	**Default:** REXXLIBLEN=NO_REXXLIBLEN
,UTOKEN=*utoken*	*utoken:* RS-type address or address in register (2) - (12)
,UTOKEN=TASK	**Default:** UTOKEN=TASK
,TSO=NO	**Default:** TSO=NO
,TSO=YES	
,REXXINDSN=*rexxindsn*	*xrexxindsn*: RS-type address or address in register (2) - (12)
,REXXINDSN=NO_REXXINDSN	**Default:** REXXINDSN=NO_REXXNDSN
,REXXINEMEMNAME=*rexxinmemname*	*rexxinmemname:* RS-type address or address in register (2) - (12)
,REXXINMEMNAME=NO_REXXINMEMNAME	**Default:** REXXINMEMNAME=NO_REXXINMEMNAME
,CONSDATA=NO	**Default:** CONSDATA=NO
,CONSDATA=YES	

Syntax	Description
,CART=*cart*	*cart:* RS-type address or address in register (2) - (12)
,CONSNAME=*consname*	*consname:* RS-type address or address in register (2) - (12)
,TIMELIMIT=YES	**Default:** TIMELIMIT=YES
,TIMELIMIT=NO	
,TIMEINT=*timeint*	*timeint:* RS-type address or address in register (2) - (12)
,TIMEINT=SYSTEM	**Default:** TIMEINT=SYSTEM
,NAME=*name*	*name:* RS-type address or address in register (2) - (12)
,REXXARGS=*rexxargs*	*rexxargs:* RS-type address or address in register (2) - (12)
,REXXARGS=NO_ARGS	**Default:** REXXARGS=NO_ARGS
,REXXVARS=*rexxvars*	*rexxvars:* RS-type address or address in register (2) - (12)
,REXXVARS=NO_VARS	**Default:** REXXVARS=NO_VARS
,REXXOUTDSN=*rexxoutdsn*	*rexxoutdsn:* RS-type address or address in register (2) - (12)
,REXXOUTDSN=NO_REXXOUTDSN	**Default:** REXXOUTDSN=NO_REXXOUTDSN
,REXXOUTMEMNAME=*rexxoutmemname*	*rexxoutmemname:* RS-type address or address in register (2) - (12)
,REXXOUTMEMNAME=NO_REXXOUTMEMNAME	**Default:** REXXOUTMEMNAME=NO_REXXOUTMEMNAME
,REXXDIAG=*rexxdiag*	*rexxdiag:* RS-type address or address in register (2) - (12)
,SYNC=YES	**Default:** SYNC=YES
,SYNC=NO	

Syntax	Description
,OREQTOKEN=*oreqtoken*	*oreqtoken:* RS-type address or address in register (2) - (12)
,REQTOKEN=*reqtoken*	*reqtoken:* RS-type address or address in register (2) - (12)
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12)
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12)
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

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

REQUEST=EXECUTE
REQUEST=CANCEL
REQUEST=GETREXXLIB
A required parameter, which identifies the request type.

REQUEST=EXECUTE
Executes a REXX exec.

REQUEST=CANCEL
Cancels a prior Execute request.

REQUEST=GETREXXLIB
Returns the REXXLIB concatenation.

,SECURITY=BYUTOKEN
,SECURITY=BYAXRUSER
When REQUEST=EXECUTE is specified, an optional parameter that indicates how the security environment should be established for the exec. The default is SECURITY=BYTOKEN.

,SECURITY=BYUTOKEN
Keyword that indicates that the security environment should be established from the UTOKEN that was passed or defaulted.

,SECURITY=BYAXRUSER
Keyword that indicates that the security environment should be established from the value of AXRUSER specified in AXR00.

,REXXLIB=*xrexxlib*
When REQUEST=GETREXXLIB is specified, a required parameter that indicates the storage area where the REXXLIB concatenation details are returned. For the mapping of this storage area, see AxrRxlHeader and AxrRxlEntry of AXRZARG in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).

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

,REXXLIBLEN=*xrexxliblen*
When REQUEST=GETREXXLIB is specified, a required parameter containing the length of the value of the Rexxlib parameter in bytes. The length must be greater than or equal to 20480.

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

,UTOKEN=*utoken*
,UTOKEN=TASK
When SECURITY=BYUTOKEN and REQUEST=EXECUTE are specified, an optional keyword that contains the security token to be used to establish the security environment under which the exec is to be executed. The optional input field contains the address of the security token to be associated with the execution of the REXX exec. The REXX exec will run under the security environment associated with the input UTOKEN. Additionally, if the REXX exec invokes the AXRCMD function, the UTOKEN is passed to MGCRE to provide security information. You can obtain the UTOKEN value by using the RACROUTE REQUEST=TOKENXTR, RACROUTE REQUEST=VERIFYX, or RACROUTE REQUEST=TOKENBLD macros. See *z/OS Security Server RACROUTE Macro Reference* for more information about the RACROUTE macros. The UTOKEN should be that of the user on whose behalf the exec is issued. UTOKEN is an optional parameter; if it is omitted, the UTOKEN of the invoker will be used. The default is TASK, which indicates the use of the UTOKEN associated with the task invoking AXREXX.

To code: Specify the RS-type address, or address in register (2) - (12), of an 80-character field.

,TSO=NO
,TSO=YES
When REQUEST=EXECUTE is specified, an optional parameter that indicates whether the exec is to be run in a TSO host command environment. If the exec is to perform dynamic allocation (e.g. with TSO ALLOCATE or BPXWDYN), it should be run in the TSO=YES environment. The default is TSO=NO.

,TSO=NO
Indicates that the exec is to run in an MVS host command environment, in an address space with up to 63 other concurrently running execs.

,TSO=YES
Indicates the exec is to be run in a TSO host command environment. In this case, the exec will run isolated in a separate address space with no other concurrent work. Not all of the services and functionality of TSO will be present. Additionally, TSO services which depend upon JES as the primary subsystem will not work. See *z/OS MVS Programming: Authorized Assembler Services Guide* for a discussion of what TSO services are supported. TSO=YES users should be aware that there is a limit of 8 TSO Server address spaces.

,REXXINDSN=*rexxindsn*
,REXXINDSN=NO_REXXINDSN
When TSO=NO and REQUEST=EXECUTE are specified, an optional input parameter containing the name of the data set that the PARSE external function will read data from. The exec may obtain the DDNAME associated with this data set by accessing the AXRINDD variable. The default is NO_REXXINDSN.

To code: Specify the RS-type address, or address in register (2) - (12), of a 44-character field.

,REXXINMEMNAME=*rexxinmemname*
,REXXINMEMNAME=NO_REXXINMEMNAME
When REXXINDSN=rexxindsn, TSO=NO and REQUEST=EXECUTE are specified, an optional input parameter containing the name of the member in the data set specified by the REXXINDSN keyword. The default is NO_REXXINMEMNAME.

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

,CONSDATA=NO
,CONSDATA=YES
When REQUEST=EXECUTE is specified, an optional keyword that indicates whether the results of the execution of the exec are to be treated as a system command.

,CONSDATA=NO
Indicates that the exec is not being invoked as a system command.

,CONSDATA=YES
Indicates that the exec is invoked as a system command. It also specifies console attributes of the issuer to be used on AXRWTO or AXRMLWTO function invocations that the REXX exec may make.

,CART=*cart*
When CONSDATA=YES and REQUEST=EXECUTE are specified, the address of an 8-character field that contains the name of the command and response token to be used on any AXRWTO or AXRMLWTO invocations by the exec.

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

,CONSNAME=*consname*
When CONSDATA=YES and REQUEST=EXECUTE are specified, the name of the console to be used with any AXRWTO or AXRMLWTO function invocations in the EXEC.

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

,TIMELIMIT=YES
,TIMELIMIT=NO
When REQUEST=EXECUTE is specified, an optional parameter indicating whether a time limit is applied. This time limit does not include the time the request spends waiting to be dispatched.

,TIMELIMIT=YES
Indicates that a time limit should be applied.

,TIMELIMIT=NO
Indicates that no time limit is to be applied.

,TIMEINT=*timeint*
,TIMEINT=SYSTEM
When TIMELIMIT=YES and REQUEST=EXECUTE are specified, you can specify an optional input parameter containing the number of seconds to allow the REXX exec to run. If the exec exceeds the threshold, it will be stopped and a return or reason code will be set indicating so. A maximum of 21474536 seconds can be specified. A value of 0 is equivalent to TIMELIMIT=NO. The default is SYSTEM, which indicates that a default of 30 seconds will be used.

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

,NAME=*name*
When REQUEST=EXECUTE is specified, a required input parameter containing the name of the REXX exec.

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

,REXXARGS=*rexxargs*
,REXXARGS=NO_ARGS
When REQUEST=EXECUTE is specified, you can specify an optional input/output parameter containing the argument list to be passed to the REXX program. The mapping of the argument list

is specified by a header section mapped by AXRARGLST followed by one or more sections mapped by AXRARGENTRY for each argument. The entries mapped by AXRARGENTRY must appear in the same order as the arguments specified on the ARG statement in the REXX program. The mappings for both AXRARGLST and AXRARGENTRY can be found in AXRZARG. For more detailed information about how to initialize this parameter see *z/OS MVS Programming: Authorized Assembler Services Guide*. The default is NO_ARGS.

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

,REXXVARS=*rexxvars*
,REXXVARS=NO_VARS
When REQUEST=EXECUTE is specified, you can specify an optional input/output parameter containing a variable list that can be used to initialize variables in the REXX programs. The variable list can also be used to obtain the final value of variables in the REXX program. Use the same mapping as RexxArgs. The default is NO_VARS.

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

,REXXOUTDSN=*rexxoutdsn*
,REXXOUTDSN=NO_REXXOUTDSN
When REQUEST=EXECUTE is specified, an optional input parameter containing the name of the data set that the exec will direct the output from SAY, error messages and tracing to. The REXX exec may obtain the DDNAME associated with this data set by accessing the AXROUTDD variable. The default is NO_REXXOUTDSN.

To code: Specify the RS-type address, or address in register (2)-(12), of a 44-character field.

,REXXOUTMEMNAME=*rexxoutmemname*
,REXXOUTMEMNAME=NO_REXXOUTMEMNAME
When REXXOUTDSN=rexxoutdsn and REQUEST=EXECUTE are specified, an optional input parameter containing the name of the member in the data set specified by the REXXOUTDSN keyword. The default is NO_REXXOUTMEMNAME.

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

,REXXDIAG=*rexxdiag*
When REQUEST=EXECUTE is specified, an optional output parameter buffer containing the return code from the exec and diagnostic data. For SYNC=NO invocations, not all the diagnostic data from the execution of the exec will be returned. In particular, the return code from the exec will not be returned. See AXRDIAG in AXRZARG for a mapping.

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

,SYNC=YES
,SYNC=NO
When REQUEST=EXECUTE is specified, you can specify an optional parameter that indicates whether the request is synchronous. The default is SYNC=YES.

,SYNC=YES
Indicates the request is synchronous.

,SYNC=NO
Indicates the request is asynchronous.

,OREQTOKEN=oreqtoken
When SYNC=NO and REQUEST=EXECUTE are specified, an optional output parameter containing a unique token associated with this EXECUTE request.

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

,REQTOKEN=reqtoken
When REQUEST=CANCEL is specified, a required input parameter containing the token that was returned when the EXECUTE request was made.

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

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=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 the same value on all of the macro forms used for a request. 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**, indicates you want the parameter list to be the largest size currently possible. This size might grow from release to release and affect the amount of storage that your program needs.

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

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

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

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

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

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

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

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

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

ABEND codes

None.

Return and reason codes

When the AXREXX 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 AXRZARG 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 20. Return and reason codes for the AXREXX macro

Return code	Reason code	Equate symbol, meaning and action
0	—	**Equate symbol:** AxrRetCodeOK **Meaning:** AXREXX request successful. **Action:** None required. If RexxDiag was specified, AXRDIAG1, AXRDIAG2, AXRDIAG3, and AXRDIAG4 contain the message ids of any message beginning with IRX (REXX) or IKJ (TSO) that were issued when processing the exec. The format of the message id is packed decimal with the sign bits shifted out. A 1 in the high order byte distinguishes an IKJ message from and IRX message.
8	—	**Equate symbol:** AxrRetcodeError **Meaning:** The AXREXX request failed due to a user error. **Action:** Refer to the action provided with the specific reason code.
8	xxxx0801	**Equate symbol:** AXRNoFrrAllowed **Meaning:** Caller invoked AXREXX with an FRR. **Action:** Remove the FRR and then invoke AXREXX.
8	xxxx0802	**Equate symbol:** AXRNoLocksAllowed **Meaning:** Caller invoked AXREXX holding a lock. **Action:** Free the lock and then invoke AXREXX.
8	xxxx0803	**Equate symbol:** AXRNotTcbMode **Meaning:** Caller was not running as a task. **Action:** Move the invocation of AXREXX under a task.
8	xxxx0804	**Equate symbol:** AXRNotAuthorized **Meaning:** Caller is not APF authorized, running in a system key or in supervisor state. **Action:** Avoid invoking AXREXX in this environment.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx0805	**Equate symbol:** AXRNotEnabled **Meaning:** Caller is disabled. **Action:** Avoid invoking AXREXX in this environment.
8	xxxx0806	**Equate symbol:** AXRRexxArgsCannotAccess **Meaning:** The RexxArgs parameter is not accessible. **Action:** Verify that the RexxArgs parameter is accessible and in the key in which AXREXX was invoked
8	xxxx0807	**Equate symbol:** AXRArgCannotAccess **Meaning:** An argument in the argument list cannot be accessed. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter to determine the index of the argument that was not accessible. Verify that AxrArgAddr and AxrArgAlet contain the address and alet of the argument. Verify that the argument is in the same key as the invoker.
8	xxxx0808	**Equate symbol:** AxrArgBadLength **Meaning:** The length of an argument is not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter to determine the index of the argument whose length was incorrect. Correct AxrArgLength.
8	xxxx0809	**Equate symbol:** AxrArgBadType **Meaning:** Type of an argument is not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter to determine the index of the argument whose type is incorrect. Correct AxrArgType with one of the valid types listed in AXRZARG.
8	xxxx080A	**Equate symbol:** AXRPlistCannotAccess **Meaning:** The input parameter list was not accessible. **Action:** Verify that the input parameter list is in the same key as the invoker. Verify that it is accessible.
8	xxxx080B	**Equate symbol:** AxrArgTooMany **Meaning:** Too many arguments were specified. **Action:** Verify the contents of AxrArgLstNumber in the RexxArgs parameter. The maximum possible value is 20.
8	xxxx080C	**Equate symbol:** AxrArgBadNumeric **Meaning:** The output argument from a REXX exec is not numeric. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter for the index of the invalid argument. Make sure that the REXX exec did not return a value in scientific notation.
8	xxxx080D	**Equate symbol:** AXRArgBadBitString **Meaning:** The output argument from a REXX exec is not a bit string. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter for the incorrect argument. Correct the exec or change AxrArgType.
8	xxxx080E	**Equate symbol:** AXRArgBadHexString **Meaning:** The output argument from a REXX exec is not a hex string. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter for the index of the incorrect argument. Correct the exec or change AxrArgType.
8	xxxx0810	**Equate symbol:** AXRArgBadNameLength **Meaning:** The length of the name of an argument is too long. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter for the index of the incorrect argument. Correct AxrArgNameLength.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx0811	**Equate symbol:** AXRNotAbleToAllocateRexxInDsn **Meaning:** The REXX processor was unable to allocate the REXXINDsn data set. **Action:** The return and reason codes from DYNALLOC are inserted into AXRDIAG1 and AXRDIAG2 in the RexxDiag parameter. Look up the return and reason codes in the *z/OS MVS Programming: Authorized Assembler Services Guide*. Look in the System Log for any messages that were issued by DYNALLOC.
8	xxxx0812	**Equate symbol:** AXRNotAbleToAllocateRexxOutDsn **Meaning:** The System REXX processor was unable to allocate the RexxOutDsn data set. **Action:** The return and reason codes from DYNALLOC are inserted into AXRDIAG1 and AXRDIAG2 in the RexxDiag parameter. Look up the return and reason codes in the *z/OS MVS Programming: Authorized Assembler Services Guide*. Look in the System Log for any messages that were issued by DYNALLOC.
8	xxxx0813	**Equate symbol:** AXRUtokenCannotAccess **Meaning:** Unable to access the Utoken input parameter. **Action:** Ensure that the Utoken input parameter is in the key of the AXREXX invoker and that it is accessible.
8	xxxx0814	**Equate symbol:** AXRRexxInDsnCannotAccess **Meaning:** Unable to access the RexxInDsn input parameter. **Action:** Ensure that the RexxInDsn input parameter is in the key of the AXREXX invoker and that it is accessible.
8	xxxx0815	**Equate symbol:** AXRRexxOutDsnCannotAccess **Meaning:** Unable to access the RexxOutDsn input parameter. **Action:** Ensure that the RexxOutDsn input parameter is in the key of the AXREXX invoker and that it is accessible.
8	xxxx0816	**Equate symbol:** AXRRexxVarsCannotAccess **Meaning:** Unable to access the RexxVars input parameter. **Action:** Ensure that the RexxVars parameter is accessible and in the key in which AXREXX was invoked.
8	xxxx0817	**Equate symbol:** AXRBadTimeInt **Meaning:** The value of the Timeint keyword is not valid. **Action:** Ensure that the value of the TimeInt keyword is less than 21474536 seconds.
8	xxxx0818	**Equate symbol:** AXRArgBadAcronym **Meaning:** The acronym for the RexxArgs keyword is incorrect. **Action:** Ensure that AxrArgLstID is set to AxrArgLstAcro.
8	xxxx0819	**Equate symbol:** AXRVarBadAcronym **Meaning:** The acronym for the RexxVars keyword is not correct. **Action:** Ensure that AxrArgLstId is set to AxrVarLstAcro.
8	xxxx081A	**Equate symbol:** AXRArgBadVersion **Meaning:** The version for the RexxArgs keyword is not correct. **Action:** Ensure that the version is one that is supported.
8	xxxx081B	**Equate symbol:** AXRVarBadVersion **Meaning:** The version for the RexxVars keyword is not correct. **Action:** Ensure that the version is one that is supported.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx081C	**Equate symbol:** AxrVarTooMany **Meaning:** Too many variables were specified. **Action:** Verify the contents of AxrArgLstNumber in the RexxVars parameter. The maximum possible value is 256.
8	xxxx081D	**Equate symbol:** AxrVarBadNumeric **Meaning:** An output variable from a REXX exec is not numeric. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter for the index of the incorrect variable. Make sure that the REXX exec did not return a value in scientific notation.
8	xxxx081E	**Equate symbol:** AXRVarBadBitString **Meaning:** An output variable from a REXX exec is not a bit string. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter for the index of the incorrect variable. Correct the exec or change AxrArgType.
8	xxxx081F	**Equate symbol:** AXRVarBadHexString **Meaning:** An output variable from a REXX exec is not a hex string. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter for the index of the incorrect variable. Correct the exec or change AxrArgType.
8	xxxx0820	**Equate symbol:** AXRVarBadNameLength **Meaning:** The length of the name of a variable is too long. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter for the index of the incorrect variable. Correct the AxrArgNameLength.
8	xxxx0821	**Equate symbol:** AXRVarBadType **Meaning:** The type specification for a variable is not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter for the index of the incorrect variable. Correct AxrArgType with one of the valid types listed in AXRZARG.
8	xxxx0822	**Equate symbol:** AXRVarCannotAccess **Meaning:** A variable could not be accessed. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter for the index of the variable that could not be accessed. Ensure that AxrArgAddr and AxrArgAlet contain the address and alet of the variable. Ensure that the variable is in the same key as the invoker.
8	xxxx0823	**Equate symbol:** AXRVarBadLength **Meaning:** The length of a variable was not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter for the index of the variable whose length is not valid. Correct AxrArgLength.
8	xxxx0824	**Equate symbol:** AXRArgLstRsvNotZero **Meaning:** A reserved field in the AXRARGLST mapping was non-zero for the RexxArgs AXREXX parameter. **Action:** Clear the reserved fields in the AXRARGLST mapping.
8	xxxx0825	**Equate symbol:** AXRVarLstRsvNotZero **Meaning**: A reserved field in the AXRARGLST mapping was not zero for the RexxVars AXREXX parameter. **Action**: Clear the reserved fields in the AXRARGLST mapping.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx0826	**Equate symbol:** AXRNotAbleToUnallocateRexxInDsn **Meaning**: A bad return code was returned from DYNALLOC when attempting to unallocate the RexxinDsn data set. **Action**: The return and reason codes from DYNALLOC are insterted into AXRDIAG1 and AXRDIAG2 in the RexxDiag parameter. Look in the System Log for any messages that DYNALLOC may have issued.
8	xxxx0827	**Equate symbol:** AXRNotAbleToUnallocateRexxOutDsn **Meaning**: A bad return code was returned from DYNALLOC when attempting to unallocate the RexxOutDsn data set. **Action**: The return and reason codes from DYNALLOC are inserted into AXRDIAG1 and AXRDIAG2 in the RexxDiag parameter. Look in the System Log for any messages that DYNALLOC may have issued.
8	xxxx0828	**Equate symbol:** AXRExecSyntaxError **Meaning**: A syntax error or another run time error was encountered during the execution of a REXX exec. **Action**: The REXX interpreter issues one or more error messages that indicate the offending line number. If RexxOutDsn is specified, look at the data set for the message. If RexxOutDsn is not specified but CONSDATA is specified, look at the console or the system log. If RexxDiag is specified: • AXRDIAG1 contains the number of the error which corresponds to an IRXnnnI message. • AXRDIAG2 contains the line number where the error occurred. • AXRDIAG3 and AXRDIAG4 contain the message ids of the last two IRX or IKJ messages that were issued before the exec completed. All of these values are in packed decimal format with the sign bits shifted out. A 1 in the high order byte distinguishes an IKJ message from an IRX message.
8	xxxx082A	**Equate symbol:** AXRArgNumericTooBig **Meaning**: The value of an output argument was either too large or too small to be represented in the buffer that was passed. **Action**: Inspect AxrArgLstEntryInError in the RexxArgs parameter for the index of the argument that caused the error.
8	xxxx082B	**Equate symbol:** AXRVarNoExist **Meaning**: An output variable was not set in the exec. **Action**: Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the output variable that caused the error. Determine why this variable was not set in the exec.
8	xxxx082C	**Equate symbol:** AXRArgNoExist **Meaning**: An output argument was not set in the exec. **Action**: Inspect AxrArgLstEntryInError in the RexxArgs parameter for the index of the argument that caused the error. Determine why this argument was not set in the exec.
8	xxxx082D	**Equate symbol:** AXRVarTooLong **Meaning**: The buffer of the client could not accommodate the value of the variable. **Action**: Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the variable that caused the error. Increase the size of the output variable or ensure that the variable's size can be accommodated by the passed buffer.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx082E	**Equate symbol:** AXRArgTooLong **Meaning**: The buffer of the client could not accommodate the value of the argument. **Action**: Inspect AxrArgLstEntryInError in the RexxArgs parameter for the index of the argument that caused the error. Increase the size of the output argument or ensure that the argument's size can be accommodated by the passed buffer.
8	xxxx082F	**Equate symbol:** AXRVarBadName **Meaning**: The name of a variable was not acceptable to REXX. **Action**: Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the variable that caused the error. Correct the name.
8	xxxx0830	**Equate symbol:** AXRArgBadName **Meaning**: The name of an argument was not acceptable to REXX. **Action**: Inspect AxrArgLstEntryInError in the RexxArgs parameter for the index of the argument that caused the error. Correct the name.
8	xxxx0831	**Equate symbol:** AXRVarNumericTooBig **Meaning**: A value of an output variable was either too large or too small to be represented in the buffer that was passed. **Action**: Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the variable that caused the error.
8	xxxx0832	**Equate symbol:** AXRArgNameCannotAccess **Meaning**: The argument name was not accessible. **Action**: Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the argument that caused the error. Ensure that AxrArgNameAddr and AxrArgNameAlet contain the address and alet of the argument name. Ensure that the argument name is in the same key as the invoker.
8	xxxx0833	**Equate symbol:** AXRVarNameCannotAccess **Meaning**: The variable name was not accessible and caused a program check when System REXX attempted to access. **Action**: Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the variable that caused the error. Ensure that AxrArgNameAddr and AxrArgNameAlet contain the address and alet of the variable name. Ensure that the variable name is in the same key as the invoker.
8	xxxx0835	**Equate symbol:** AXRDiagCannotAccess **Meaning**: The value of the RexxDiag parameter was not accessible. **Action**: Ensure the RexxDiag parameter is in the same key as the invoker. Correct the RexxDiag parameter.
8	xxxx0837	**Equate symbol:** AXRArgNeitherInOrOut **Meaning**: A REXX argument is neither an input or output argument. **Action**: Inspect AxrArgLstEntryInError in the RexxArgs parameter for the index of the offending argument and set either AXRArgInput, AXRArgOutput or both in the argument list entry.
8	xxxx0838	**Equate symbol:** AXRVarNeitherInOrOut **Meaning**: A REXX variable is neither an input or output variable. **Action**: Inspect AxrArgLstEntryInError in the RexxArgs parameter for the index of the offending variable and set either AXRArgInput, AXRArgOutput, or both in the entry in the variable list entry.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx0839	**Equate symbol:** AXRArgBadUnsigned **Meaning**: An unsigned argument returned from REXX was prefixed with a sign. **Action**: AxrArgLstEntryInError in the RexxArgs parameter contains the index of the invalid argument. Correct the REXX exec to return an unsigned value or change the argument to signed.
8	xxxx083A	**Equate symbol:** AXRVarBadUnsigned **Meaning**: An unsigned variable returned from REXX was prefixed with a sign. **Action**: AxrArgLstEntryInError in the RexxVars parameter contains the index of the invalid variable. Change the exec to return an unsigned value or change the variable to be signed.
8	xxxx083B	**Equate symbol:** AXRBadConsoleName **Meaning**: The specified CONSNAME parameter was syntactically incorrect. **Action**: Correct the syntax of the CONSNAME parameter so that it is a syntactically valid console name.
8	xxxx083E	**Equate symbol:** AXRRexxInNotAuth **Meaning**: Invoker is not SAF authorized to the data set name specified on the RexxInDsn keyword. **Action**: Either change the data set name or change the security environment so that the data set can be accessed.
8	xxxx083F	**Equate symbol:** AXRRexxOutNotAuth **Meaning**: Invoker is not SAF authorized to access the data set name specified on the RexxOutDsn keyword. **Action**: Either change the data set name or change the security environment so that the data set can be accessed.
8	xxxx0840	**Equate symbol:** AXRRexxInDsnBad **Meaning:** The RexxInDsn specification is not syntactically correct. **Action:** Change the input to a valid data set name.
8	xxxx0841	**Equate symbol:** AXRRexxOutDsnBad **Meaning:** The RexxOutDsn specification is not syntactically correct. **Action:** Change the input to a valid data set name.
8	xxxx0842	**Equate symbol:** AXRRacrouteBad **Meaning:** RACROUTE VERIFY returned a bad return code when attempting to create a security environment prior to running the REXX exec. **Action:** The SAF return code is stored in AXRDIAG1. The RACF® return and reason codes are stored in AXRDIAG2 and AXRDIAG3 respectively (all in the REXXDiag parameter). Certain types of address spaces do not have a legitimate security environment and as such the AXREXX invoker may have to provide a different UTOKEN or use SECURITY=BYAXRUSER.
8	xxxx0843	**Equate symbol:** AXRRexxOutCannotOpen **Meaning:** A failure occurred when attempting to open the data set specified by RexxOutDsn. **Action:** The return code from IRXINOUT is set in AXRDIAG1. The return code is documented in *z/OS TSO/E REXX Reference*. Additionally, the REXX interpreter may issue messages describing the error.
8	xxxx0844	**Equate symbol:** AXRRexxInCannotOpen **Meaning:** A failure occurred when attempting to open the specified by RexxInDsn. **Action:** The return code from IRXINOUT is set in AXRDIAG1 in the RexxDiag parameter and is documented in the *z/OS TSO/E REXX Reference*. Additionally, the REXX interpreter may issue messages describing the error.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx0846	**Equate symbol:** AXRBadRequest **Meaning:** The AXREXX input parameter list is incorrect. An incorrect request type is specified. **Action:** Determine why the AXREXX input parameter list is incorrect.
8	xxxx0847	**Equate symbol:** AXRArgRsvNotZero **Meaning:** A reserved field in the AXRARGENTRY mapping was non-zero for the RexxArgs AXREXX parameter. **Action:** AxrArgLstEntryInError in the RexxArgs parameter contains the index of the entry that caused the error. Clear the reserved fields.
8	xxxx0848	**Equate symbol:** AXRVarRsvNotZero **Meaning:** A reserved field in the AXRARGENTRY mapping was non-zero for the RexxVars AXREXX parameter. **Action:** AxrArgLstEntryInError in the RexxVars parameter contains the index of the entry that caused the error. Clear the reserved fields.
8	xxxx0849	**Equate symbol:** AXRBadReqToken **Meaning:** For a CANCEL request, the input Request Token is not valid. **Action:** Correct the invocation to provide a valid Request Token.
8	xxxx084A	**Equate symbol:** AXRRexxInNotSeq **Meaning:** RexxInDsn is a PDS, but RexxInMemName is not specified. **Action:** Specify the RexxInMemName keyword or change RexxInDsn.
8	xxxx084B	**Equate symbol:** AXRRexxInNotPDS **Meaning:** RexxInMemName is specified but RexxInDsn is not a PDS. **Action:** Remove RexxInMemName or specify a PDS for RexxInDsn.
8	xxxx084C	**Equate symbol:** AXRRexxOutNotSeq **Meaning:** RexxOutDsn is a PDS, but RexxOutMemName is not specified. **Action:** Specify the RexxOutMemName keyword or change RexxOutDsn.
8	xxxx084D	**Equate symbol:** AXRRexxOutNotPDS **Meaning:** RexxOutMemName is specified but RexxOutDsn is not a partitioned data set (PDS). **Action:** Remove the RexxOutMemName keyword or change the specification of RexxOutDsn to a PDS.
8	xxxx084E	**Equate symbol:** AXRRexxInNotMember **Meaning:** RexxInMemName does not exist in the data set specified by RexxInDsn. **Action:** Either create the member or specify a different RexxInDsn data set name.
8	xxxx0850	**Equate symbol:** AXRVarBadValue **Meaning:** The value of an input variable was not acceptable to REXX. **Action:** Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the variable that caused the error.
8	xxxx0851	**Equate symbol:** AXRExecNotFound **Meaning:** The exec was not found in the System REXX library. **Action:** Correct the spelling of the exec in the NAME keyword.
8	xxxx0852	**Equate symbol:** AXRVarOutBadValue **Meaning:** The value of an output variable was not acceptable to REXX. **Action:** Inspect AxrArgLstEntryInError in the RexxVars parameter for the index of the variable that caused the error.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx0853	**Equate symbol:** AXRArgOutBadValue **Meaning:** The value of an output argument was not acceptable to REXX. **Action:** Inspect AxrARgLstEntryInError in the RexxArgs parameter for the index of the argument that caused the error.
8	xxxx0854	**Equate symbol:** AXRParmlistBadAlet **Meaning:** The ALET of the parmlist is not valid. **Action:** Correct the Alet.
8	xxxx0855	**Equate symbol:** AXRUtokenBadAlet **Meaning:** The ALET of the UTOKEN parameter is not valid. **Action:** Correct the Alet.
8	xxxx0856	**Equate symbol:** AXRRexxArgsBadAlet **Meaning:** The ALET of the REXXARGS parameter is not valid. **Action:** Correct the Alet.
8	xxxx0857	**Equate symbol:** AXRRexxVarsBadAlet **Meaning:** The ALET of the REXXVARS parameter is not valid. **Action:** Correct the Alet.
8	xxxx0858	**Equate symbol:** AXRRexxInDsnBadAlet **Meaning:** The ALET of the REXXINDSN parameter is not valid. **Action:** Correct the Alet.
8	xxxx0859	**Equate symbol:** AXRRexxOutDsnBadAlet **Meaning:** The ALET of the REXXOUTDSN parameter is not valid. **Action:** Correct the Alet.
8	xxxx085A	**Equate symbol:** AXRRexxDiagBadAlet **Meaning:** The ALET of the REXXDIAG parameter is not valid. **Action:** Correct the Alet.
8	xxxx085B	**Equate symbol:** AXRArgBadAlet **Meaning:** The ALET of the argument entry is not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter to determine the index of the argument entry whose alet was incorrect. Correct AxrArgAlet.
8	xxxx085C	**Equate symbol:** AXRArgNameBadAlet **Meaning:** The ALET of the argument entry name is not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxArgs parameter to determine the index of the argument entry name whose alet was incorrect. Correct AxrArgNameAlet.
8	xxxx085D	**Equate symbol:** AXRVarBadAlet **Meaning:** The ALET of the variable entry is not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter to determine the index of the variable entry whose alet was incorrect. Correct AxrArgAlet.
8	xxxx085E	**Equate symbol:** AXRVarNameBadAlet **Meaning:** The ALET of the variable entry name is not valid. **Action:** Refer to AxrArgLstEntryInError in the RexxVars parameter to determine the index of the variable entry name whose alet was incorrect. Correct AxrArgNameAlet.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
8	xxxx085F	**Equate symbol:** AXRRexxlibBadAlet **Meaning:** The ALET of the Rexxlib parameter is not valid. **Action:** Correct the ALET.
8	xxxx0860	**Equate symbol:** AXRBadRexxlibLen **Meaning:** The length specified by the RexxlibLen keyword is not valid. **Action:** Ensure that the specified RexxlibLen is greater than or equal to 8192.
8	xxxx0861	**Equate symbol:** AXRBadRexxlib **Meaning:** A program check occurred when attempting to access the parameter specified by the REXXLIB keyword. **Action:** Correct the Rexxlib keyword.
C	—	**Equate symbol:** AxrRetcodeEnvError **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	xxxx0C01	**Equate symbol:** AxrNotActive **Meaning**: Function is not available. Either the AXR address space has terminated or has not initialized. **Action**: Avoid requesting this function until the ENF signal for AXR initialization is issued or message AXR0102I is issued. If the AXR address space has terminated, it needs to be restarted.
C	xxxx0C02	**Equate symbol:** AxrArgNoStorage **Meaning**: No storage is available for a REXX argument or variable. **Action**: Reissue the AXREXX request after the requests that are in progress complete.
C	xxxx0C03	**Equate symbol:** AXRAllReqBlocksInUse **Meaning**: All the storage available to represent REXX requests is in use. **Action**: Reissue the AXREXX request after the requests that are in progress complete.
C	xxxx0C04	**Equate symbol:** AXRTooManyRexxReqs **Meaning**: The threshold of active and waiting REXX requests has been exceeded. **Action**: System REXX will issue ENF signal (65) with a qualifier of '10000000'x to indicate that it has begun accepting new requests. The AXREXX invoker can listen for this signal.
C	xxxx0C05	**Equate symbol:** AXRBadIWMEREG **Meaning**: A bad return code was returned from IWMEREG. The return code and reason codes from IWMEREG are placed in AXRDIAG1 and AXRDIAG2 in the RexxDiag parameter respectively. **Action**: Examine the return and reason codes from IWMEREG. If no diagnosis is possible, contact IBM Service.
C	xxxx0C06	**Equate symbol:** AXRAscreFailed **Meaning:** An attempt to create a server address space to run the exec failed. **Action:** The return and reason codes from ASCRE are stored in AxrDiag1 and AxrDiag2 in the RexxDiag parameter.
C	xxxx0C07	**Equate symbol:** AXRReqCancelled **Meaning:** The request was cancelled. **Action:** None.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
C	xxxx0C08	**Equate symbol:** AXRExecRexxEnvError **Meaning:** The REXX Interpreter was unable to run the exec. **Action:** The REXX Interpreter issues one or more messages describing the problem. If REXXOUTDSN was specified, look in the data set for the messages. If CONSDATA was specified and REXXOUTDSN was not specified, look at the console or the system log. If RexxDiag was specified, AXRDIAG1, AXRDIAG2, AXRDIAG3 and AXRDIAG4 contain the message ids of any messages beginning with IRX (REXX) or IKJ (TSO) that were issued. The format of the message id is packed decimal with the sign bits shifted out. A 1 in the high order byte distinguishes an IKJ message from an IRX message.
C	xxxx0C09	**Equate symbol:** AXRBadAxruser **Meaning:** AXRUSER was improperly defined in parmlib member AXR00. **Action:** Correct AXR00 and restart System REXX.
C	xxxx0C0A	**Equate symbol:** AXRTimeIntExpired **Meaning:** The input time limit expired before the exec completed. **Action:** Increase the time limit or modify the exec.
C	xxxx0C0B	**Equate symbol:** AXRReqNotActive **Meaning:** For a CANCEL request, the request to be cancelled is no longer active. **Action:** None.
C	xxxx0C0C	**Equate symbol:** AXRReqAlreadyCancelled **Meaning:** For a CANCEL request, the request to be cancelled is already cancelled. **Action:** None.
C	xxxx0C0D	**Equate symbol:** AXRRexxOutFail **Meaning:** A failure occurred when attempting to process the data set specified by the RexxOutDsn parameter. If the failure was due to an abend, the abend code is saved in AXRDIAG1 and the abend reason code is saved in AXRDIAG2 in the RexxDiag parameter. No dump is taken. **Action:** Look up the abend and reason code in the *z/OS MVS System Codes* to determine the proper action.
C	xxxx0C0E	**Equate symbol:** AXRRexxInFail **Meaning:** A failure occurred when attempting to process the data set specified by the RexxInDsn parameter. If the failure was due to an abend, the abend code is saved in AXRDIAG1 and the abend reason code is saved in AXRDIAG2 of the RexxDiag parameter. No dump is taken. **Action:** Look up the abend and reason code in *z/OS MVS System Codes* to determine the proper action.
C	xxxx0C0F	**Equate symbol:** AXRBadIWMECREA **Meaning:** A bad return code was returned from IWMECREA. The return code and reason codes from IWMESQRY and the return and reason codes from IWMECREA are placed in AXRDIAG1, AXRDIAG2, AXRDIAG3 and AXRDIAG4 respectively of the RexxDiag parameter. **Action:** Inspect the return and reason codes from IWMESQRY and IWMECREA by looking it up in the *z/OS MVS Programming: Workload Management Services*. If the problem cannot be resolved, contact IBM Service.
C	xxxx0C10	**Equate symbol:** AXRsTooManyExtents **Meaning:** The number of extents for all data sets in the REXXLIB concatenation exceeds the system limit. SYSREXX cannot process any more execs. **Action:** SYSREXX must be stopped. Modify the REXXLIB concatenation so that the number of extents is reduced below the system limit. You can then restart SYSREXX.

Table 20. Return and reason codes for the AXREXX macro (continued)

Return code	Reason code	Equate symbol, meaning and action
C	xxxx0C11	**Equate symbol:** AXRStopTSO **Meaning:** AXREXX TSO=YES processing is stopped. The MODIFY AXR,SYSREXX STOPTSO command has been issued and any subsequent AXREXX TSO=YES requests will be rejected. **Action:** To resume processing of AXREXX TSO=YES requests, issue MODIFY AXR,SYSREXX STARTTSO from the operator console.
0Cx	xxxx0C12	**Equate symbol:** AXRStopInProgress **Meaning:** The STOP AXR command was invoked and is being processed. **Action:** The request is rejected. System REXX must be restarted before AXREXX can be invoked.
10	—	**Equate symbol:** AxrRetcodeCompError **Meaning**: Unexpected failure. **Action**: Refer to the action provided with the specific reason code.
10	xxxx1001	**Equate symbol:** AxrRexxServerAbended **Meaning**: An abend occurred after the REXX server began processing the request. **Action**: Contact IBM Service with information from the dump.
10	xxxx1002	**Equate symbol:** AxrBadServerRC **Meaning**: An unexpected return code was returned from the REXX server. **Action**: Contact IBM Service with information from the dump.
10	xxxx1003	**Equate symbol:** AXRRexxClientAbended **Meaning**: An abend occurred before the request was passed to the REXX Server or after the request was processed by the REXX server. **Action**: Contact IBM Service with information from the dump.
10	xxxx1007	**Equate symbol:** AXRExitAbended **Meaning**: An abend occurred in a System REXX defined exit, which is given control by the REXX interpreter or in the REXX interpreter. **Action**: If the abend occurred within a System REXX exit, contact IBM Service with information from the dump.
10	xxxx100B	**Equate symbol:** AXRAddrSpaceTerm **Meaning:** The address space created to run an exec either terminated before running the exec or while running the exec. **Action:** If the address space was cancelled then there is no action. If the address space was terminated unexpectedly, then contact IBM Service.
10	xxxx100D	**Equate symbol:** AXRCancelAbended **Meaning:** An attempt to cancel a request resulted in an abend. **Action:** Contact IBM Service with information from the dump.
10	xxxx100F	**Equate symbol:** AXRRexxInterpreterAbend **Meaning:** Either the REXX interpreter abended or was percolated to. **Action:** See the RexxDiag parameter. AxrDiag1 contains either 100 for a user abend or 104 for a system abend. AxrDiag2 contains the abend code. A system dump may be produced.

Examples

See *z/OS MVS Programming: Authorized Assembler Services Guide* for examples.

Chapter 12. AXSET — Set authorization index

Description

The AXSET macro sets the authorization index (AX) of the home address space to the value specified by the caller. The AX must be reserved. The address space in which the AX is being changed cannot own SASN=OLD connected space switch entry tables. All routines that subsequently execute, with a PASID of the address space for the AX being changed, execute with the new AX.

Related macros

ATSET, AXFRE, AXRES, and AXEXT

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM = 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	Any
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Primary

Programming requirements

None.

Restrictions

None.

Input register information

The AXSET 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 AXSET 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

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

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

Register
Contents

0
Bits 0-15 contain zeros; bits 16-31 contain the replaced Authority Index (AX)

1
Used as a work register by the macro

2-13
Unchanged

14
Used as a work register by the macro

15
Return code

Performance implications

None.

Syntax

This is the standard form of the AXSET macro:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede AXSET.
AXSET	
ƀ	One or more blanks must follow AXSET.
AX=*AX value*	*AX value*: RX-type address or register (0) - (12).
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

These are the parameters:

AX=*AX value*
Specifies the new AX value. The RX-type address specifies a halfword containing the new AX. When the register form is used, the register must contain the new AX in bits 16-31, and bits 0-15 must be zero.

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

ABEND codes

- 052
- 053

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

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

Table 21. Return Code for the AXSET Macro

Hexadecimal Return Code	Meaning
00	The AX of the home address space is set to the value specified by the caller.

Examples

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

Chapter 13. BCPii — Base control program internal interface services

IBM provides support within z/OS that allows authorized applications to query, change, and perform basic operational procedures against the installed System z hardware base. This support provides a set of high-level application programming interfaces (APIs) for data exchange and command requests. The functions, called base control program internal interface (BCPii), is delivered in the base of the operating system. This support not only allows control of the hardware that the APIs are executing on, but also extends to other System z processors within the attached process control network. This support does not require communication on an IP network for connectivity to the support element (SE) / hardware management console (HMC). Calls using the BCPii APIs can be made from either C or assembler programming languages.

The services listed below are all documented in *z/OS MVS Programming: Callable Services for High-Level Languages*. See that information for a complete description of the BCPii APIs as well as an overall description of how to use BCPii on the z/OS operating system.

That information contains the following callable services:

- HWICMD Callable Service
- HWICONN Callable Service
- HWIDISC Callable Service
- HWIEVENT Callable Service
- HWILIST Callable Service
- HWIQUERY Callable Service
- HWISET Callable Service
- HWIBeginEventDelivery Callable Service
- HWIEndEventDelivery Callable Service
- HWIManageEvents Callable Service
- HWIGetEvent Callable Service

Chapter 14. BPXEKDA — Kernel data access

Description

The BPXEKDA macro provides an interface for an authorized report application to retrieve kernel data, such as:

- A list of current processes (for the entire system or for a specific userid)
- A list of threads within a specific process
- A list of current BPXPRMxx option settings
- All system-wide file system information

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	Unlocked
Control parameters:	All parameters must be addressable by the caller and in the primary address space.

Programming requirements

If the supplied buffer is in the primary address space (ALET=0), BPXEKDA will protect the last page of the buffer to guard against overlays of storage following the buffer. If the buffer is not in the primary address space, the caller is responsible for protecting the last page of the buffer to prevent potential overlays.

Restrictions

None.

Input register information

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

Register
Contents

0-12
Undefined

13
Address of a 72-byte register save area

14-15
Undefined

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Unpredictable

2-13
Unchanged

14
Next instruction address

15
Return code

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: symbol. The name is optional.
ƀ	One or more blanks must precede BPXEKDA.
BPXEKDA	
ƀ	One or more blanks must follow BPXEKDA.
,KBUFLEN=*xkbuflen*	*xkbuflen*: RS-type address or register (2) - (12).
,KBUFALET=*xkbufalet*	*xkbufalet*: RS-type address or register (2) - (12).
KBUFPTR=*xkbufptr*	*xkbufptr*: RS-type address or register (2) - (12).
,RETCODE=*xretcode*	*xretcode*: RS-type address or register (2) - (12).
,MF=S	**Default:** MF=S

Parameters

The parameters are explained as follows:

name
An optional symbol, starting in column 1, that is the name on the BPXEKDA macro invocation. The name must conform to the rules for an ordinary assembler language symbol. **Default**: No name.

,KBUFLEN=*xkbuflen*
The name (RS-type) or address in register (2) - (12) of a required fullword field that contains the length of the supplied buffer. It is recommended that this buffer be at least 1 megabyte in length to contain all of the data that could be returned for the input request data. Upon successful completion, this area is defined by the mapping macro BPXZODMV. If BPXEKDA is unsuccessful, this area contains incorrect data.

,KBUFALET=*xkbufalet*
The name (RS-type) or address in register (2) - (12) of a required fullword field that contains the ALET of the supplied buffer.

KBUFPTR=*xkbufptr*
The name (RS-type) or address in register (2) - (12) of a required 4–byte area that contains the address of a required input/output area that is to contain the input request data and the output data to be returned by BPXEKDA. The mapping macro BPXZODMV maps all of this input and output data. The input request data is mapped by the OdmvInputParms field. The remainder of the BPXZODMV mapping macro describes data that is returned by BPXEKDA when it completes successfully.

,RETCODE=xretcode
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.

,MF=S
An optional parameter that requests the standard form, which places parameters into an inline parameter list and invokes the BPXEKDA macro service. **MF=S** is the default.

Return codes

The following are the return codes from BPXEKDA and their explanations:

Table 22. Return Codes for the BPXEKDA Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: BPXEKDA completed successfully. **Action:** None required.
04	**Meaning**: The macro was unsuccessful. The supplied user name or ASID was not found. **Action**: None required.
08	**Meaning**: The macro was unsuccessful. An internal error occurred attempting to obtain file system mount data. **Action**: None required.
12	**Meaning**: The macro was unsuccessful. The supplied buffer was not large enough to hold all of the data that the macro attempted to return. **Action**: None required.

Table 22. Return Codes for the BPXEKDA Macro (continued)	
Hexadecimal Return Code	**Meaning and Action**
16	**Meaning**: The macro was unsuccessful. The input flags specified mutually exclusive values. **Action**: None required.
20	**Meaning**: The macro was unsuccessful. The caller does not have the correct authorization to use this service. **Action**: None required.
24	**Meaning**: The macro was unsuccessful. The caller did not specify the address of an output area in OdmvOutPtr. **Action**: None required.
28	**Meaning**: OMVS inactive **Action**: None required.

BPXEKDA - List form

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

Syntax	Description
name	*name*: symbol. The name is optional.
ƀ	One or more blanks must precede BPXEKDA.
BPXEKDA	
ƀ	One or more blanks must follow BPXEKDA.
,MF=(L,*xmfctrl*)	*xmfctrl*: Symbol.
,MF=(L,*xmfattr*)	*xmfattr*: 1– to 60–character input string. **Default**: 0D.

Parameters

The parameters of the list form are explained as follows:

,MF=(L,*xmfctrl*)
MF=(L,*xmfctrl* ,*xmfattr*
Specifies the list form of the BPXEKDA macro.

xmfctrl is the name of a storage area to contain the parameters.

xmfattr 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 *xmfattr*, the system provides a value of 0D, which forces the parameter list to a doubleword boundary.

BPXEKDA - Execute form

Syntax

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

Syntax	Description
name	*name*: symbol. The name is optional.
ƀ	One or more blanks must precede BPXEKDA.
BPXEKDA	
ƀ	One or more blanks must follow BPXEKDA.
,KBUFLEN=*xkbuflen*	*xkbuflen*: RS-type address or register (2) - (12).
,KBUFALET=*xkbufalet*	*xkbufalet*: RS-type address or register (2) - (12).
KBUFPTR=*xkbufptr*	*xkbufptr*: RS-type address or register (2) - (12).
,RETCODE=*xretcode*	*xretcode*: RS-type address or register (2) - (12).
,MF=(E,*xmfctrl*)	*xmfctrl*: RS-type address or register (2) - (12).

Parameters

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

,MF=(E,*xmfctrl*)
Specifies the execute form of the BPXEKDA macro.

xmfctrl specifies the area that the system uses to store the parameters.

Chapter 15. BPXESMF — Collect z/OS UNIX process accounting data

Description

The BPXESMF macro provides z/OS UNIX accounting data when the macro targets an address space that is an z/OS UNIX process.

The caller provides the address of a storage area where process accounting data is to be written after successful completion of BPXESMF. The contents of this storage area are defined by mapping macro BPXYOSMF. For a multiprocess address space, the statistics that are returned represent the sum of the services that each process in the address space consumed. The process and session identifiers are the ones from the first process.

SMF recording must be active for type 30 records in order for some of the values returned in BPXYOSMF to be accumulated.

Environment

The requirements for the caller are:

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

Programming requirements

The program issuing the BPXESMF macro should include the mapping macro, BPXYOSMF. Before invoking BPXESMF, a storage area of size, OSMF#LENGTH, should be allocated to contain macro output.

After invoking BPXESMF, the return code should be checked to verify that the macro completed successfully. If the macro did not complete successfully, output should be discarded as incorrect.

OSMFVERSION value returned will be the lesser of the OSMF#VERSION passed via the BPXESMF invocation and OSMF#VERSION used during the compilation of the system service.

OSMFLENGTH value will identify the last field touched by the system service and corresponds to OSMF#VERSION found in the appropriate level of BPXYOSMF.

Restrictions

None.

Input register information

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

Register
Contents

13
Address of a 72-byte register save area.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Next instruction address

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

There are no performance implications.

Syntax

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

Syntax	Description
name	*name*: symbol. The name is optional.
ƀ	One or more blanks must precede BPXESMF.
BPXESMF	
ƀ	One or more blanks must follow BPXESMF.
ACCTDAT=*acctdata*	*acctdata*: RX-type address or register (2) - (12).
,ASCBPTR=*ascbptr*	*ascbptr*: RX-type address or register (2) - (12).

Syntax	Description
,RETCODE=*rc*	*rc*: RX-type address or register (2) - (12).
,MF=S	**Default:** MF=S

Parameters

The parameters are explained as follows:

ACCTDAT=*acctdata*
Specifies the location of the storage area where BPXESMF is to place output process accounting data.

,ASCBPTR=*ascbptr*
Specifies the location of the address space control block for the address space from which accounting data is to be collected. Specifying ASCBPTR=PSAAOLD is requesting accounting data for the current process and is equivalent of not specifying this keyword.

A program that uses IHAAVST to cycle through all address spaces, should construct a loop (1 to AvstMaxU) and invoke BPXESMF for each valid (AvstAval(i) = OFF) address space (AvstEnty(i)).

,RETCODE=*rc*
Specifies the location where the system is to store the return code. The return code is also in GPR 15. RETCODE is an optional parameter.

,MF=S
An optional parameter that requests the standard form, which places parameters into an inline parameter list and invokes the BPXESMF macro service. MF=S is the default.

ABEND codes

Users of ASCBPTR should be able to recover from an occasional 0C4 program check because collection of accounting data will be made without protection that the target address space may be deleted.

Return codes

When the BPXESMF macro returns control to your program, GPR 15 contains a return code. No reason code is returned by this macro.

Table 23. Return Codes for the BPXESMF Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: BPXESMF completed successfully **Action**: The storage area passed in parameter ACCTDAT can now be mapped using macro BPXYOSMF.
08	**Meaning**: The macro was unable to obtain process accounting data. **Action**: Normally this means that the primary address space is not an z/OS UNIX process when the macro is invoked. The contents of the storage area passed in parameter ACCTDAT should be discarded as invalid.
0C	**Meaning**: No longer used. **Action**: Apply the proper PTF to BPXAMSMF or recompile with the BPXESMF macro of the level of the system.

Example

The following example shows how a program running in supervisor state, key zero, might invoke the BPXESMF macro to obtain z/OS UNIX process accounting data for the primary address space. Key steps in this program are as follows:

1. Assign register 13 to the address of a 72-byte register save area. In this example it is assumed that standard linkage is used on entry. Therefore, register 13 is assumed to already be pointing to a standard 72-byte register save area.
2. Issue the GETMAIN macro to obtain storage for the process accounting data. The size of the storage area needed can be found in the OSMF#LENGTH equate in macro BPXYOSMF.
3. Issue the BPXESMF macro to obtain process accounting data.
4. Verify successful completion of the macro before using the output process accounting data.
5. If BPXESMF returned successfully, the BPXYOSMF macro is used to map output process accounting data.

```
*
*    GET STORAGE FOR PROCESS ACCOUNTING DATA
*
         LA    R0,OSMF#LENGTH
         GETMAIN  RU,LV=(R0)
         LR    R9,R1
         USING OSMF,R9
*
*    ISSUE BPXESMF TO GET PROCESS ACCOUNTING DATA
*
         BPXESMF ACCTDAT=OSMF
         LTR   R15,R15
         BNZ   ERROR
*
*    USE VALID PROCESS ACCOUNTING DATA
*
         MVC   UID,OSMFRUID
         MVC   PID,OSMFPROCESSID
         . . .
         . . .
         B     DONE
*
*    HANDLE BPXESMF ERRORS
*
ERROR    EQU   *
         . . .
         . . .
DONE     LR    R1,R9
         LA    R0,OSMF#LENGTH
         FREEMAIN RU,LV=(R0),A=(R1)
         . . .
         . . .
@SA00001 DS    18F
UID      DS    F
PID      DS    F
         BPXYOSMF
```

BPXESMF - List form

Use the list form of the BPXESMF macro together with the execute form of the macro for applications that require reentrant code. The list form of the macro defines an area of storage, which the execute form of the macro uses to contain the parameters.

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

Syntax	Description
name	*name*: symbol. Begin *name* in column 1. The name is optional.

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

The parameters of the list form are explained as follows:

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

BPXESMF - Execute form

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede BPXESMF.
BPXESMF	
ƀ	One or more blanks must follow BPXESMF.
ACCTDAT=*acctdata*	*acctdata*: RX-type address or register (2) - (12).

Syntax	Description
,ASCBPTR=*ascbptr*	*ascbptr*: RX-type address or register (2) - (12).
,RETCODE=*rc*	*rc*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or address in register (2) - (12).

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

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

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

Chapter 16. CALLDISP — Pass control to another ready task

Description

The CALLDISP macro saves the caller's status in the current TCB/RB, and passes control to another ready task. The task with the highest priority is the one that receives control. When the original task is re-dispatched, control is returned to the next sequential instruction.

Environment

These are the requirements for the caller:

- When BRANCH=NO

Environmental factor	Requirement
Minimum authorization:	None.
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:	None.

- When BRANCH=YES

Note: When BRANCH=YES, the caller must include the IHAPSA mapping macro.

Environmental factor	Requirement
Minimum authorization:	When FIXED=NO: Supervisor state or PKM allowing key 0 When FIXED=YES: Supervisor state and PSW key 0
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN or PASN¬=HASN
AMODE:	24- or 31-bit
ASC mode:	Any
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	None.

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the CALLDISP 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

0-13
Unchanged

14
Unchanged when BRANCH=NO, used as a work register by the system when BRANCH=YES

15
Used as a work register by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as a work register by the system

Performance implications

None.

Syntax

This is the standard form of the CALLDISP macro:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CALLDISP.
CALLDISP	
ƀ	One or more blanks must follow CALLDISP.
BRANCH=NO	**Default:** BRANCH=NO

Syntax	Description
BRANCH=YES	
,FIXED=YES	**Default:** (Available only when BRANCH=YES is coded)
,FIXED=NO	FIXED=YES
,FRRSTK=SAVE	**Default:** (Available only when BRANCH=YES is coded)
,FRRSTK=NOSAVE	FRRSTK=NOSAVE

Parameters

These are the parameters:

BRANCH=NO
BRANCH=YES
Specifies whether the branch entry (BRANCH=YES) or the SVC entry (BRANCH=NO) of CALLDISP is to be used. The default is BRANCH=NO.

BRANCH=YES is restricted to key 0 supervisor state callers. Routines in cross memory mode must specify BRANCH=YES. See *z/OS MVS Programming: Authorized Assembler Services Guide* for more information about the requirements for using the BRANCH=YES option of the CALLDISP Macro.

Routines that are unlocked, have no enabled unlocked task FRRs on the stack, and are not in cross memory mode, can use BRANCH=NO.

,FIXED=YES
,FIXED=NO
Specifies that the code invoking branch entry CALLDISP is in fixed storage (FIXED=YES) or in pageable storage (FIXED=NO). For FIXED=NO, registers 14-1 are altered.

,FRRSTK=SAVE
,FRRSTK=NOSAVE
Specifies that the current FRR stack be saved and restored (FRRSTK=SAVE), when at least one of the FRRs is an enabled unlocked task (EUT) FRR, or purged (FRRSTK=NOSAVE). When FRRSTK=SAVE is specified:

- The caller cannot hold any locks or an abend results.
- When EUT FRRs exist, the current FRR stack is saved and the caller can hold either the LOCAL or CML lock.
- When no EUT FRR exists, the caller cannot hold any locks. Otherwise, an abend occurs.
- Asynchronous exits (IRBs and SIRBs) are not dispatched until all EUT FRRs are deleted.

For more information, see “Suspension and Resumption of Request Blocks” in *z/OS MVS Programming: Authorized Assembler Services Guide* for an explanation of the CALLDISP function used with SUSPEND/RESUME processing.

Abend codes

05D

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return and reason codes

None.

Example 1

Pass control to another ready task.

```
CALLDISP
```

Example 2

A non-page-fixed task with an enabled, unlocked task FRR gives control to another ready task. When the original task regains control, the contents of registers 14, 15, 0, and 1 have been destroyed.

```
CALLDISP  FIXED=NO,FRRSTK=SAVE,BRANCH=YES
```

Chapter 17. CALLRTM — Call recovery termination manager

Description

The CALLRTM macro schedules abnormal termination for a task, an address space, or a preemptive SRB identified by the token returned by the SRBIDTOKEN= parameter of the IEAMSCHD macro.

To terminate a task, use TYPE=ABTERM and the following parameter or parameters to identify the specific task.

- For a task in the home address space, use TCB or TTOKEN.
- Use one of the following for a task in a specific address space:
  - TTOKEN and ASID
  - TCB and ASID.

To terminate an address space, use TYPE=MEMTERM. However, be aware that tasks in the abending address space cannot perform the recovery and task-level resource managers do not get control.

Note: Address space recovery routines and resource managers do get control.

To terminate an address space, consider using CALLRTM TYPE=ABTERM,RETRY=NO to abend each job step task in the address space. When all the tasks in the address space terminate, the system terminates the address space.

To terminate a preemptive SRB identified by the token returned by the SRBIDTOKEN= parameter of the IEAMSCHD macro, use TYPE=SRBTERM. The target SRB is processed asynchronously and might terminate after control returns to the invoking program. Also, the target SRB can finish normally before RTM can terminate it with the requested completion code. If the target SRB has not started, it can be purged instead of abended (in which case its RMTR receives control).

For more information about using CALLRTM, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	For TYPE=MEMTERM and TYPE=SRBTERM, any PASN, any HASN, any SASN. For TYPE=ABTERM, see the TCB and TTOKEN parameter descriptions.
AMODE:	24 or 31 bit, except for TYPE=SRBTERM, which supports only AMODE 31 callers.
ASC mode:	Primary or secondary
Interrupt status:	When using TTOKEN to terminate a task other than the current one, the caller must not be disabled for I/O and external interrupts.
Locks:	When terminating a task without specifying ASID, the caller must hold the LOCAL lock. When terminating an SRB, any lock can be held, but none are required.

Environmental factor	Requirement
Control parameters:	For callers in primary ASC mode, control parameters must be in the primary address space; in secondary mode, control parameters must be in the secondary address space.

Programming requirements

- When the caller runs in 31-bit addressing mode, all input parameters except the TCB can reside above 16 MB. The TCB always resides below 16 MB.
- The caller must include the CVT mapping macro.

Restrictions

None.

Input register information

Before issuing the CALLRTM macro with TYPE=MEMTERM, or with TYPE=ABTERM with ASID or TTOKEN or both, the caller must ensure that the following general-purpose registers (GPRs) contain the specified information.

Register
Contents

13
The address of a 72-byte work area.

Note: The work area that you provide is not the standard 72-byte save area. The system stores into the area. If you pass in register 13 the save area that you are using to link your program to your caller's, you will not be able to get back to your caller.

Before issuing the CALLRTM macro with TYPE=SRBTERM, the caller must ensure that the following general-purpose registers (GPRs) contain the specified information.

Register
Contents

13
The address of a 144-byte work area.

Note: The work area that you provide is not the standard 144 bytes save area. The system stores into the area. If you pass in register 13 the save area that you are using to link your program to your caller's, you will not be able to get back to your caller.

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
If you specify the ASID parameter, which is used as a work register by the system; otherwise, unchanged.

3
If you specify the DUMPOPT or DUMPOPX parameter, which is used as a work register by the system; otherwise, unchanged.

4-5
Unchanged

6
If you specify the TYPE=ABTERM or TYPE=SRBTERM, COMPCOD, and REASON parameters, which are used as a work register by the system; otherwise, unchanged.

7-13
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 which remain the same before and after issuing a service. If the system changes the contents of registers 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 CALLRTM macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CALLRTM.
CALLRTM	
ƀ	One or more blanks must follow CALLRTM.
TYPE=ABTERM	
TYPE=MEMTERM	
TYPE=SRBTERM	
,SRBIDTOKEN=*token*	*token*: RS-type address, or address in register (2) - (12).

Syntax	Description
,COMPCOD=*comp code*	*comp code*: Symbol, decimal digit, or register (1) - (12).
,REASON=*code*	*code*: Symbol, decimal or hexadecimal number, or register (2) - (12).
,ASID=*asid*	*asid*: Decimal digits 0-32,765 or register (2) - (14) or, when TTOKEN is not specified, register (15).
,TCB=*tcb addr*	*tcb addr*: 0, or register (2) - (12).
,TTOKEN=*ttoken*	*ttoken*: 0, RS-type address, or address in register (2) - (12).
	Note: Use TCB and TTOKEN only with TYPE=ABTERM.
	Default: TCB=0
,STEP=NO	**Default:** STEP=NO
,STEP=YES	**Note:** Use STEP only with TYPE=ABTERM.
,DUMP=YES	**Default:** DUMP=YES
,DUMP=NO	**Note:** Use DUMP only with TYPE=ABTERM.
,DUMPOPT=*parm list addr*	*parm list addr*: Address in register (3) - (14) or, when TTOKEN is not specified, address in register (15).
,DUMPOPX=*parm list addr*	*parm list addr*: Address in register (3) - (14) or, when TTOKEN is not specified, address in register (15).
	Note: Use DUMPOPT and DUMPOPX only with DUMP=YES.
,RETRY=YES	**Default:** RETRY=YES
,RETRY=NO	**Note:** Use RETRY only with TYPE=ABTERM.
,SYSTEM=YES	**Default:** SYSTEM=YES
,SYSTEM=NO	

Parameters

The parameters are explained as follows:

TYPE=ABTERM
TYPE=MEMTERM
TYPE=SRBTERM,SRBIDTOKEN=*token*
Specifies whether CALLRTM is to terminate a task (ABTERM), an address space (MEMTERM), or a preemptive SRB (SRBTERM) identified by the token returned by the SRBIDTOKEN= parameter of the IEAMSCHD macro.

For TYPE=ABTERM, you must identify the task through the TCB or TTOKEN parameters.

For TYPE=MEMTERM, no task-level recovery processing occurs.

For TYPE=SRBTERM, *token* is a 16-byte token that uniquely identifies the preemptive SRB to be terminated, including its ASID. The following CALLRTM parameters are valid when TYPE=SRBTERM has been specified: COMPCOD, REASON, RETRY, SYSTEM. The default values for RETRY and SYSTEM are YES. COMPCOD must be specified. Use CALLRTM TYPE=SRBTERM only when the SRBIDTOKEN has been successfully returned by the IEAMSCHD macro, as indicated by a nonzero value in the first 8 bytes of the SRBIDTOKEN.

,COMPCOD=*compcode*
Specifies the system (if you specify SYSTEM=YES or take the default) or user (if you specify SYSTEM=NO) completion code (0 - 4096 decimal) that you associate with the abnormal termination. Specify this parameter as a hexadecimal code (x'80A'), a decimal code (2058), or a register contains a hexadecimal code. In all cases, the result is hexadecimal.

,REASON=*code*
Specifies information to supplement the completion code associated with an abnormal termination. The value range for the reason code is a 32-bit hexadecimal number or 31-bit decimal number. In all cases, the result is hexadecimal.

The system passes the reason code value to the recovery routine in the SDWACRC field of the SDWA.

For TYPE=SRBTERM, set the high-order bit of the reason code to 1 when you want to indicate that an SVCDUMP is not necessary for the abend being issued. The system does not do anything special with this bit, but recovery routines can use the information when determine if they request an SVCDUMP of the abend.

,ASID=*asid*
Specifies the address space to be terminated (for MEMTERM), or the one that contains the task to be terminated (for ABTERM). ASID=0, the default, specifies the home address space.

,TCB=*tcb addr*
,TTOKEN=*ttoken*
Specifies the task to be terminated. TCB=0 is the default, which identifies the current task.

tcb addr is the address of the TCB that CALLRTM is to terminate.

If the current task is specified or implicitly specified by not specifying either TCB= or TTOKEN=, a nonzero ASID parameter must also be specified.

When you specify TCB=*tcb address* and you omit the ASID parameter, the system assumes the task is in the home address space and that the home address space is addressable. That is:

- If you are in primary ASC mode, the primary address space must be the home address space (PASN=HASN).
- If you are in secondary ASC mode, the secondary address space must be the home address space (SASN=HASN).

ttoken specifies the TTOKEN for the task that is to be terminated. ASID with TTOKEN identifies a task in the specified address space.

When you omit the ASID parameter, there are requirements on locks. See "Environment" on page 151.

,STEP=NO
,STEP=YES
Specifies whether the job step task associate with the specified task is (YES) or is not (NO) to be abnormally terminated if the specified task terminates.

Note: The job step task does not end abnormally if the specified task successfully retries.

STEP is valid only for TYPE=ABTERM.

,DUMP=YES
,DUMP=NO
Specifies whether a dump is (YES) or is not (NO) to be taken. You can use DUMPOPT or DUMPOPX to specify the dump options; otherwise, the contents of the dump are defined by the //SYSABEND, // SYSMDUMP, or //SYSUDUMP DD statement and the system or user-defined defaults. The target address space of the CALLRTM request is treated as the dump error address space.

The final decision on whether a dump is taken depends on the recovery routines that run as a result of this CALLRTM. If the recovery routines indicate in the ",DUMP="option of the SETRP macro whether a dump is to be taken, this specification overrides the ",DUMP="value in CALLRTM.

,DUMPOPT=*parm list addr*
,DUMPOPX=*parm list addr*
Specifies the address of a parameter list of dump options. To create the parameter list, use the list form of the SNAP or SNAPX macro; or build the parameter list by coding your own data constants. DUMPOPT specifies the address of a parameter list that the SNAP macro creates. DUMPOPX specifies the address of a parameter list that the SNAPX macro creates. When you terminate a task that is not the current one, the dump options must reside in fixed or disabled reference (DREF) storage.

The system dump options, which are specified by the CHNGDUMP operator command, can add to or override this parameter list. All recovery routines entered for the failure can also add to the list of dump options. The TCB, DCB, ID, and STRHDR options available on SNAP or SNAPX are ignored when they appear in the parameter list. The TCB is for the task that receives the ABEND. The DCB is provided by the ABDUMP routine. When a //SYSABEND, //SYSMDUMP, or //SYSUDUMP DD statement is not provided, the system ignores the DUMPOPT or DUMPOPX parameters.

Note: When you use this parameter, the system ruins the contents of register 3.

,RETRY=YES
,RETRY=NO
Specifies whether the target task's recovery routines can retry. If you specify RETRY=NO, the recovery routines are forced to percolate rather than retry. RETRY is valid only for TYPE=ABTERM. RETRY=YES is the default.

,SYSTEM=YES
,SYSTEM=NO
Specifies whether the completion code is to be a system or user completion code.

ABEND codes

CALLRTM might abnormally terminate with abend code X'70D'. See *z/OS MVS System Codes* for an explanation of this abend code and its associated reason codes.

Return codes

When CALLRTM returns control to your program, for TYPE=ABTERM, register 15 contains one of the following hexadecimal return codes.

Table 24. Return codes for the CALLRTM macro for TYPE=ABTERM

Hexadecimal return code	Meaning and action
00	**Meaning:** The ABTERM request was processed successfully. **Action:** None.
04	**Meaning:** The task has already been scheduled for termination by a previous ABTERM request. **Action:** None.
08	**Meaning:** An asynchronous unit of work has been scheduled to terminate the task. **Action:** None.
18	**Meaning:** Program error. The ASID value is not valid. **Action:** Ensure that the ASID specified represents a currently active address space.

Table 24. Return codes for the CALLRTM macro for TYPE=ABTERM (continued)	
Hexadecimal return code	**Meaning and action**
1C	**Meaning:** Program error. The TCB address or TTOKEN does not represent a valid TCB. **Action:** Ensure that the TCB address or TTOKEN represents a valid TCB within the primary address space or the ASID parameter is also specified to further qualify the target TCB.
20	**Meaning:** Program error. TTOKEN specifies a TCB in another address space. **Action:** The ASID parameter must also be specified to terminate the TCB in another address space.
24	**Meaning:** Program error. The caller tried to terminate a task other than the current task, but did not hold the LOCAL lock. **Action:** Ensure that the LOCAL lock is obtained before issuing the CALLRTM macro for this type of request.
28	**Meaning:** Program error. TTOKEN specifies a task other than the current one and the caller is disabled for I/O and external interrupts. **Action:** Ensure that the caller is enabled for I/O and external interrupts before issuing the CALLRTM macro for this type of request.

When CALLRTM returns control to your program, for TYPE=MEMTERM, register 15 contains one of the following hexadecimal return codes.

Table 25. Return codes for the CALLRTM macro for TYPE=MEMTERM	
Hexadecimal return code	**Meaning and action**
00	**Meaning:** The MEMTERM request was scheduled successfully. Termination of an address space might also be temporarily deferred by a system service. When the deferral condition is released, the system accepts the MEMTERM request. **Note:** Memory termination occurs asynchronously and that the actual termination process might not have started when the CALLRTM service returns to its caller. **Action:** None.
18	**Meaning:** Program error. The ASID value is not valid. **Action:** Ensure that the ASID represents a currently active address space.
2C	**Meaning:** Environmental error. The requested ASID represents an address space that is non-memtermable. **Action:** The memory termination request is not accepted. Depending on the circumstances involved, it might be appropriate to stop the system with a WAIT state when this return code is received.

When CALLRTM returns control to your program, for TYPE=SRBTERM, register 15 contains one of the following hexadecimal reason and return codes. The first 3 bytes of the register contain the reason code and the last byte contains the return code.

Table 26. Reason and return codes for the CALLRTM macro for TYPE=SRBTERM		
Hexadecimal reason code	**Hexadecimal return code**	**Meaning and action**
000000	00	**Meaning:** The SRBTERM request was scheduled successfully. The target SRB will be terminated at the next opportunity. **Action:** None.
000001	04	**Meaning:** The SRBIDTOKEN is no longer valid. This return code implies that the target SRB has already terminated. **Action:** None.

Table 26. Reason and return codes for the CALLRTM macro for TYPE=SRBTERM (continued)		
Hexadecimal reason code	**Hexadecimal return code**	**Meaning and action**
000002	04	**Meaning:** An SRBTERM request with RETRY=YES was issued against an SRB for which a previous SRBTERM request with RETRY=NO is still being processed. The older RETRY=NO SRBTERM is honored rather than the new RETRY=YES SRBTERM. **Action:** None.
000001	08	**Meaning:** The SRBIDTOKEN contains data that is not valid. **Action:** Ensure that the SRBIDTOKEN parameter points to a valid token, which was returned by the IEAMSCHD service.
000001	10	**Meaning:** System error. The target SRB terminates if it is running, but can not terminate if it is suspended or stopped. **Action:** If the SRB does not terminate, reissue the SRBTERM request a reasonable number of times. If the SRB still does not terminate, report this error to the appropriate IBM support personnel.
000002	10	**Meaning:** System error. The target SRB will not be terminated. **Action:** Report this error to the appropriate IBM support personnel.

Example 1

Terminate the primary address space with a completion code of 123.

```
CALLRTM   TYPE=MEMTERM,COMPCOD=123,ASID=0
```

Example 2

Schedule the TCB, addressed in register 8, for abnormal termination. The abnormal termination of the TCB takes place in the address space identified by the ASID, specified in register 5. It has a completion code of 123.

```
CALLRTM   TYPE=ABTERM,COMPCOD=123,ASID=(5),TCB=(8)
```

Example 3

Terminate the current task and its associated job step task. Register 1 identifies the completion code and register 6 identifies the accompanying reason code. Register 5 is set to the current ASID. The system does not allow the recovery routines of the job step task and its attached tasks to retry from the abend.

```
CALLRTM TYPE=ABTERM,RETRY=NO,STEP=YES,TCB=0,ASID=(5),COMPCOD=(1),REASON=(6)
```

Note: This ABTERM is processed asynchronously and control is returned to the calling program. The caller will likely want to wait on a "dummy ECB" rather than continuing processing after issuing the CALLRTM request.

Example 4

Terminate the address space identified by the contents of register 2. Register 1 identifies the completion code. TYPE=MEMTERM prevents all task-related recovery, including task resource managers, from getting control. The system gives control only to the address space's resource managers.

```
CALLRTM TYPE=MEMTERM,ASID=(2),COMPCOD=(1)
```

Chapter 18. CHANGKEY — Change virtual storage protection key

Description

The CHANGKEY macro changes the protection key and fetch protection status of one or more pages of virtual storage. CHANGKEY is valid for virtual storage that is obtained by a GETMAIN or a STORAGE macro. The storage must be obtained in page multiples.

Note: If the system cannot complete the CHANGKEY request, it restores processed pages to their initial key and fetch protection status.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space

Programming requirements

The caller must include the CVT and IHAPSA mapping macros and establish addressability to the CVT with a USING statement.

Restrictions

- CHANGKEY can be used only for storage that has been obtained using the GETMAIN or STORAGE macros
- CHANGKEY cannot be used to change the storage key to key 0
- CHANGKEY can be used only with subpools 0-127, 129–132, 203-205, 213-215, 244, 247-248, and 251-252
- All storage for which CHANGKEY is invoked must have the same initial key and fetch protection status.
- CHANGKEY cannot be used for virtual storage that has been defined as shared (through the IARVSERV macro) with a read-only or a shared-write view.

Input register information

When issuing the CHANGKEY macro, GPR 13 must point to a standard 18-word save area. If the caller is disabled, the save area must be in fixed storage.

Output register information

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

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
Return code (always 0).

When control returns to the caller, the access registers (ARs) are unchanged.

Performance implications

None.

Syntax

The CHANGKEY macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CHANGKEY.
CHANGKEY	
ƀ	One or more blanks must follow CHANGKEY.
R,BA=*page addr*,EA=*page addr*	*page addr*: A-type address or register (1) - (12).
L,LISTAD=*list addr*	**Note:** The R-type macro expansion alters the contents of register 2. EA should not be specified as (1).
	list addr: A-type address or register (1) - (12).
,KEY=*stor key*	*stor key*: Decimal digit 1-15 or register (0) or register (3) - (12).

Syntax	Description
,BRANCH=YES	

Parameters

The parameters are explained as follows:

R,BA=*page addr*,EA=*page addr*
L,LISTAD=*list addr*
Specifies the type of CHANGKEY request:

R
Indicates a request to change the key of a single area of virtual storage.

L
Indicates a request to change the key of one or more areas of virtual storage.

BA
Specifies the address of the first byte of the first page of the virtual storage area whose key is to be changed.

EA
Specifies the address of the first byte of the last page of the virtual storage area whose key is to be changed.

LISTAD specifies the address of the first doubleword of a variable length parameter list in fixed storage. The first word of each element is defined as BA above and the second word of each element as EA above. If the high-order bit of the second word is one, then that element is the last element in the parameter list.

Note:

1. BA must be less than or equal to EA.
2. BA, EA, and LISTAD are expected to be 31-bit addresses, regardless of the caller's addressing mode.

,KEY=*stor key*
Specifies the new storage key and fetch protection status for the virtual storage areas specified. If the *stor key* specification is a decimal digit, the system assumes that you want fetch protection. If you do not want fetch protection, specify the protection key in bits 24-27 of a register and leave bit 28 at zero to indicate no fetch protection.

,BRANCH=YES
The only entry available into the CHANGKEY service routine is branch entry.

ABEND codes

CHANGKEY might terminate abnormally with an abend code of X'08F'. See *z/OS MVS System Codes* for an explanation and response for this code.

Return and reason codes

CHANGKEY returns a zero return code in GPR 15.

Example 1

Change the storage key and ensure fetch protection of a single page of virtual storage addressed by register 5.

```
       L 4,FLCCVT(0,0)        LOAD ADDRESS OF THE CVT INTO REGISTER 4
       USING CVT,4            ESTABLISH ADDRESSABILITY TO CVT
       CHANGKEY R,BA=(REG5),EA=(REG5),KEY=8,BRANCH=YES
```

```
       .
       .
       CVT                  INCLUDE THE CVT
       IHAPSA               INCLUDE THE PSA
```

Example 2

Change the storage key and ensure fetch protection of two noncontiguous pages of virtual storage addressed by PAGE1 and PAGE2.

```
       L 4,FLCCVT(0,0)        LOAD ADDRESS OF THE CVT INTO REGISTER 4
       USING CVT,4            ESTABLISH ADDRESSABILITY TO CVT
       CHANGKEY L,LISTAD=PLIST,KEY=10,BRANCH=YES
         .
         .
         .
PLIST  DC 2A(PAGE1)    FIRST ELEMENT IN LIST
       DC A(PAGE2)     BA PART OF SECOND ELEMENT
       DC AL1(X'80')   INDICATES LAST ELEMENT IN LIST
       DC AL3(PAGE2)   EA PART OF SECOND ELEMENT
       CVT             INCLUDE THE CVT
       IHAPSA          INCLUDE THE PSA
```

Chapter 19. CIRB - Create interruption request block

Note: IBM recommends that you use the SCHEDIRB macro rather than CIRB.

Description

The CIRB macro initializes an interruption request block (IRB) for asynchronous exit processing.

If you intend that the IRB run under a task other than the task that issues CIRB, and you want the system to free the IRB, then you must use BRANCH=YES, having placed the address of the TCB of the task using the IRB into register 4 before issuing CIRB.

For information about asynchronous exit routines, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

These are the requirements for the caller:

- When BRANCH=NO

Environmental factor	Requirement
Minimum authorization:	None.
Dispatchable unit mode:	Task
Cross memory mode:	PASN = HASN
AMODE:	Any
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

- When BRANCH=YES

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN
AMODE:	Any
ASC mode:	Primary
Locks:	LOCAL lock held
Control parameters:	Must be in the primary address space

For BRANCH=YES:

- The caller must pass the address of the target TCB in register 4.
- The caller must include the CVT mapping macro.
- Control is returned in supervisor state, key zero, with the same lock as held on entry.

Register information

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

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

Register
Contents

0
Used as a work register by the macro

1
The address of the created IRB

2-13
Unchanged

14-15
Used as work registers by the macro

Syntax

This is the standard form of the CIRB macro:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
␣	One or more blanks must precede CIRB.
CIRB	
␣	One or more blanks must follow CIRB.
EP=*entry point addr*	*entry point addr*: RX-type address, or register (0) or (2) - (12).
,KEY=PP	**Default:** KEY=PP
,KEY=SUPR	
,MODE=PP	**Default:** MODE=PP
,MODE=SUPR	
,SVAREA=NO	**Default:** SVAREA=NO
,SVAREA=YES	

Syntax	Description
,RETIQE=YES	**Default:** RETIQE=YES
,RETIQE=NO	
,STAB=DYN	
,WKAREA=*workarea size*	*workarea size*: Decimal digit, or register (2) - (12).
	Default: zero
,BRANCH=NO	**Default:** BRANCH=NO
,BRANCH=YES	
,RETRN=NO	**Default:** RETRN=NO
,RETRN=YES	**Note:** This parameter has meaning only when RETIQE=NO is specified.
,AMODE=CALLER	**Default:** AMODE=CALLER
,AMODE=DEFINED	

Parameters

These are the parameters:

EP=*entry point addr*
Specifies the address of the entry point of the user's asynchronous exit routine.

,KEY=PP
,KEY=SUPR
Specifies whether the asynchronous exit routine operates with a key of zero (SUPR) or with a key obtained from the TCB of the task issuing the CIRB macro (PP).

,MODE=PP
,MODE=SUPR
Specifies whether the asynchronous exit routine executes in problem program (PP) or supervisor (SUPR) mode.

,SVAREA=NO
,SVAREA=YES
Specifies whether to obtain a 72-byte register save area from the virtual storage assigned to the problem program. When a save area is requested, CIRB places the save area address in the IRB. The address of this area is passed to the user routine via register 13. Do not use the SVAREA=YES parameter with the MODE=SUPR parameter or the KEY=SUPR parameter if the exit is to be invoked from a user address space task. Note that the system does not require the exit routine to save and restore registers. For exiting purposes, the asynchronous exit routine can return via the address in input register 14 or by branching to CVTEXIT.

,RETIQE=YES
,RETIQE=NO
Specifies whether the associated queue elements are request queue elements (YES) or interruption queue elements (NO).

,STAB=DYN
Specifies that the IRB (including the work area) is to be freed on termination of the exit routine.

Note: When the STAB parameter is omitted from the CIRB macro, the IRB remains available for later use by the task issuing the macro.

,WKAREA=*workarea size*
Specifies the size, in doublewords, of the work area to be included in the IRB. The area can be used to build IQEs. The first four bytes of the obtained work area contain the address of the next available IQE (RBNEXAV field). The maximum size is 255 doublewords. Note that CIRB does not clear the workarea. For example, to request an IQE but no additional workarea, specify 3, for 3 doublewords, to request a 16-byte IQE plus additional space used by MVS.

,BRANCH=NO
,BRANCH=YES
Specifies whether branch linkage (YES) or SVC linkage (NO) to CIRB is provided.

,RETRN=NO
,RETRN=YES
Specifies whether the IQE is (YES) or is not (NO) kept so it can be used again after when the asynchronous exit terminates.

,AMODE=CALLER
,AMODE=DEFINED
Specifies the addressing mode where the exit routine is to be given control.

When CALLER is specified, the exit routine receives control in the same addressing mode as the caller.

When DEFINED is specified, the addressing mode of the exit routine is pointer defined. The addressing mode is determined by the setting of the high order bit of the *entry point address* for the exit routine. When the bit is set, the addressing mode is 31-bit; when the bit is not set, the addressing mode is 24-bit.

Abend codes

None.

Return and reason codes

None.

Example 1

Create an IRB to be used in scheduling an asynchronous exit. The exit is scheduled via the IQE interface to the exit effector. It receives control in the supervisor state. The IRB is to be freed when it terminates. The exit receives control at the IQERTN label.

```
CIRB EP=IQERTN,MODE=SUPR,RETIQE=NO,STAB=DYN,BRANCH=NO
```

Example 2

Create an IRB to be used in scheduling an asynchronous exit. The RQE interface to the exit effector is used to schedule the routine. The exit gets control at the RQETEST label.

```
CIRB EP=RQETEST,KEY=SUPR,MODE=SUPR,STAB=DYN,BRANCH=NO
```

Chapter 20. CMDAUTH — Command authorization service

Description

The CMDAUTH macro verifies the RACF authorization of commands. Each parameter corresponds to a RACROUTE parameter.

There is a list and an execute form, but no standard form of the CMDAUTH macro.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and key 0 - 7
Dispatchable unit mode:	Task
Cross memory mode:	HASN=PASN=SASN
AMODE:	24- or 31-bit addressing mode
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be addressable in the caller's primary address space

Restrictions

None.

Register information

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

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

Register
Contents

0
Return code from the security product. If the security product is RACF, see the description of the return codes listed with the RACROUTE REQUEST=AUTH macro in *z/OS Security Server RACROUTE Macro Reference*.

1
Address of error messages if MSGRTN=YES is specified; otherwise, used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system

15
Return code

Programming requirements

None.

Performance implications

None.

CMDAUTH - List form

Use the list form of the CMDAUTH macro to construct a nonexecutable control program parameter list.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CMDAUTH.
CMDAUTH	
ƀ	One or more blanks must follow CMDAUTH.
,MF=(L, *cntl addr*)	*cntl addr*: RX-type address or register (2) - (12).

Parameters

The parameters for the list form of the CMDAUTH macro are explained as follows:

,MF=(L,*cntl addr*)
Specifies the list form of CMDAUTH. *cntl addr* defines the area into which the system stores the parameter list.

CMDAUTH - Execute form

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

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CMDAUTH.
CMDAUTH	
ƀ	One or more blanks must follow CMDAUTH.
ENTITY = *entity name addr*	*entity name addr*: RX-type address or register (2) - (12).
,ATTR = *access level addr*	*access level addr*: RX-type address or register (2) - (12).
,LOGSTR=*log string addr*	*log string addr*: RX-type address or register (2) - (12).
	Note: See usage note (following) for usage information.
,UTOKEN=*utoken addr*	*utoken addr*: RX-type address or register (2) - (12).
	Note: See usage note (following) for usage information.
,CNTLBLK=*cntl blk addr*	*cntl blk addr*: RX-type address or register (2) - (12).
	Note: See usage note (following) for usage information.
,CBLKTYPE=CIB	**Note:** See usage note (following) for usage information.
,CBLKTYPE=CMDX	(For use of CMDAUTH in command installation exit)
,CBLKTYPE=SSCM	
,REQSTOR = *reqstor addr*	*reqstor addr*: RX-type address or register (2) - (12).
,SUBSYS = *subsys addr*	*subsys addr*: RX-type address or register (2) - (12).
,MSGSUPP=YES	**Default:** NO
,MSGSUPP=NO	
,MSGRTN=YES	**Default:** NO
,MSGRTN=NO	

Syntax	Description
,MSGSP=*subpool number*	**Default:** 229.
,MF=(E, *cntl addr*)	*cntl addr*: RX-type address or register (2) - (12).

Usage Note: You must specify one of the following parameter combinations:

- UTOKEN and LOGSTR
- CNTLBLK and CBLKTYPE

You cannot specify both of the preceding combinations. Also note that:

- UTOKEN is not valid with CNTLBLK and CBLKTYPE
- LOGSTR is optional with CNTLBLK and CBLKTYPE
- CNTLBLK is not valid with UTOKEN and LOGSTR
- CBLKTYPE is not valid with UTOKEN and LOGSTR

You can use CNTLBLK and CBLKTYPE to obtain authorization information without having to specify the UTOKEN and LOGSTR for the command. See the description of the CBLKTYPE parameter for further information.

Parameters

The parameters are explained as follows:

ENTITY=*entity name addr*
Specifies the address of a required 39-byte input field containing the resource name for the command whose authority you are checking. If the entity name is less than 39 bytes, left-justify it and pad it on the right with blanks.

ENTITY corresponds to the RACROUTE REQUEST=AUTH parameter, ENTITY.

,ATTR=*access level addr*
Specifies the SAF access level for the command whose authority you are checking. The bits set in the 1-byte field indicate the access level. The following settings apply:

- 02 - READ
- 04 - UPDATE
- 08 - CONTROL.

ATTR corresponds to the RACROUTE REQUEST=AUTH parameter, ATTR.

LOGSTR=*log string addr*
Specifies the address of a required input field containing the command text of the command whose authority you are checking. The first byte of the input field must contain the length of the command text.

LOGSTR corresponds to the RACROUTE REQUEST=AUTH parameter, LOGSTR.

UTOKEN=*utoken addr*
Specifies the address of the UTOKEN that RACROUTE will use for command authorization.

UTOKEN corresponds to the RACROUTE REQUEST=AUTH parameter, UTOKEN.

CNTLBLK=*cntl blk addr*
Specifies the address of the control block the system passes as input to CMDAUTH.

CBLKTYPE=CIB
CBLKTYPE=SSCM
Specifies the type of control block whose address you specify on the CNTLBLK parameter.

You can use the CIB as input when you need authorization information for START, STOP, or MODIFY commands.

Use the SSCM as the control block input for any subsystems that use the CMDAUTH macro during SSI command exit (function code 10) processing.

,REQSTOR=*reqstor addr*
Specifies the address of an 8-byte character field containing the control point name. (This address identifies a unique control point within a set of control points that exists in a subsystem.) If the control point name is less than eight bytes, left-justify it and pad it on the right with blanks.

If you code this operand and RACF is installed, change the RACF router table to match the operand.

,SUBSYS=*subsys addr*
Specifies the address of an 8-byte character field containing the calling subsystem's name, version, and release level. If the subsystem's name is less than eight bytes, left-justify it and pad it on the right with blanks.

If you code this operand and RACF is installed, change the RACF router table to match the operand.

,MSGSUPP=YES
,MSGSUPP=NO
Indicates whether you want to suppress write-to-operator (WTO) messages from SAF and RACF. The default is NO.

,MSGRTN=YES
,MSGRTN=NO
Indicates whether you want CMDAUTH to return error messages to the caller. If you specify YES, CMDAUTH returns the address of the messages to register 1. The default is NO.

Note: The caller must release the storage obtained when MSGRTN = YES. The address of the message in register 1 points to the following structure. For example:

```
ST        R1,MSGPT        SAVE THE ADDRESS OF MESSAGE POINTER
USING     MSGMAP,R1       OBTAIN ADDRESSABILITY TO THE MESSAGE
...

          TITLE 'MESSAGE MAP'
MSGMAP    DSECT
          DS    0CL13
MSGHEADR  DS    0CL12     MESSAGE HEADER
MSGLEN    DS    F         LENGTH OF MESSAGE
MSGNEXTP  DS    A         ADDRESS OF NEXT MESSAGE
MSGWPL    DS    A         START OF MESSAGE WPL
MSGTXT    DS    0CL1      START OF MESSAGE TEXT
```

,MSGSP=*subpool number*
Specifies the number of the subpool into which you want error messages returned. The default is 229.

,MF=(E,*cntl addr*)
Specifies the execute form of CMDAUTH. This form generates the code to store the parameters into the parameter list and execute the CMDAUTH macro. *cntl addr* defines the area into which the system stores the parameter list.

Return codes

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

Table 27. Return Codes for the CMDAUTH Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: Command issuer is authorized to issue the command.
04	**Meaning**: No authorization decision was made.

Table 27. Return Codes for the CMDAUTH Macro (continued)	
Hexadecimal Return Code	**Meaning**
08	**Meaning**: Command issuer is not authorized to issue the command.

Example

Verify the authorization of a command. Register 4 points to the data set name and register 3 points to the access level setting.

```
DO_CMDAUTH    CMDAUTH ENTITY=(R4),ATTR=(R3),SUBSYS=SUB_NAME,
                      REQSTOR=REQ_NAME,UTOKEN=UTOKEN_ADDR,
                      LOGSTR=LOG_STR,MF=(E,CMDAUTH_LIST)
```

Chapter 21. CNZMXURF — UCME look-up service macro

Description

Use the CNZMXURF macro to locate the console control block (UCME) that contains a specific console ID. CNZMXURF can only be used to look up MCS and SMCS console IDs.

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:	The caller must hold the CMS lock.
Control parameters:	Not applicable; held in a register.

Programming requirements

Before issuing the CNZMXURF macro, place a 4-byte console ID into a register. No register save area is required; however, the input registers are saved on the linkage stack.

Restrictions

None.

Input register information

Before issuing the CNZMXURF macro, the caller must either place a 4-byte console ID into a register, or directly specify the RX-type address of a 4-byte console ID.

Output register information

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

Register
Contents

0
UCME pointer

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

0–15
Unchanged

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

Performance implications

None.

Syntax

The CNZMXURF macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CNZMXURF.
CNZMXURF	
ƀ	One or more blanks must follow CNZMXURF.
register	*register*: General purpose register Register (2-12) containing a 4-byte console ID.
or	
console-ID	*console-ID*: RX-type address containing a 4-byte console ID.
,INUSE=NO	**Default**: INUSE=NO
,INUSE=YES	

Parameters

The parameters are explained as follows:

register
Contains the 4-byte console ID for which the corresponding UCME is to be located.

console-ID
Contains the 4-byte console ID for which the corresponding UCME is to be located.

INUSE=YES
The UCME corresponding to the input console ID will be returned only if it is initialized and in use.

INUSE=NO
The UCME corresponding to the input console ID will be returned, even if it is not initialized and not in use.

ABEND codes

None.

Return and reason codes

When the CNZMXURF macro returns control to your program, register 15 contains one of the following hexadecimal return codes:

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: No errors. A pointer to the UCME containing the console ID is returned in register 0. **Action**: None.
04	**Meaning**: A UCME containing the specified console ID was not found. Register 0 contains zero. **Action**: None.
08	**Meaning**: Incorrect console ID input or there is no UCME for the specified console ID. Register 0 contains zero. • Non-MCS console class was supplied. • INTERNAL console ID (0) was supplied. • INSTREAM console ID (128) was supplied. • UNKNOWN console ID (255) was supplied. • There is no UCME for the specified 4-byte console ID or the UCME for the console ID is not initialized and in use. **Action**: None.
16	**Meaning**: CNZMXURF service is not available. **Action**: None.

Example 1

Locate the UCME associated with the 4-byte console ID in register 4.

```
CNZMXURF (4)
```

Example 2

Locate the UCME associated with the 4-byte console ID stored in field “MYCONID”.

```
CNZMXURF MYCONID
```

Chapter 22. CNZQUERY — Consoles query

Description

CNZQUERY enables you to obtain information about the consoles component. You can specify whether you want information about WTORs and the message retention facility (AMRF) returned. The information is returned in an answer area defined by mapping macro CNZMYQUA.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PKM 0
Dispatchable unit mode:	Task
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided answer area (via the ANSAREAALET parameter) must be in a 2G data space on the caller's dispatchable unit access list or must be a 2G commmon-area data space.

Programming requirements

The caller must include the CNZMYQUA macro to get a mapping of the output area which is in the data space designated by the ANSAREAALET keyword. This macro also includes symbolic constants for the return and reason codes provided by this service.

It is recommended that, after using the returned information the pages in the ANSAREAALET data space be released with the RELEASE parameter of the DSPSERV macro.

Restrictions

The caller must not have EUT FRRs established.

Input register information

Before issuing the CNZQUERY 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.

In that case, 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 if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

Performance implications

None.

Syntax

The CNZQUERY macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CNZQUERY.
CNZQUERY	
ƀ	One or more blanks must follow CNZQUERY.
WTOR=NO	DEFAULT: WTOR=NO
WTOR=YES	

Syntax	Description
,AMRF=NO	DEFAULT: AMRF=NO
,AMRF=YES	
,ANSAREAALET=*ansareaalet*	*ansareaalet:* RS-type address or address in register (2) - (12)
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

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

WTOR=NO
WTOR=YES
An optional parameter that indicates whether or not information about WTORs is to be returned.

DEFAULT: NO

WTOR=NO
Do not return information about WTORs.

WTOR=YES
Return information about WTORs. A queue of OREs is returned, each of which has field ORERWQE which contains the address of the associated WQE only when bits ORESUSP and OREINUSE are both off. When either of those bits is on, there is no associated WQE, as the building of the ORE is not yet complete. The address of the first ORE is in field CNZMYQUAH_First_ORE_Addr. The ORE is mapped by IHAORE and the address of the next ORE is in field ORELKP, with a zero value in ORELKP indicating that this is the last element of the queue. The WQE is mapped by IHAWQE. Bit Cnzmyquah_Valid_WTOR_INFO is set to 1 when the information is successfully returned.

,AMRF=NO
,AMRF=YES
A optional parameter that indicates whether or not information about the action message retention facility (AMRF) is to be returned.

DEFAULT: NO

,AMRF=NO
Do not return information about the AMRF.

,AMRF=YES
Return information about the AMRF. Three queues of WQEs are returned. The address of the first immediate action WQE is in field CNZMYQUAH_First_IA_WQE_Addr. The address of the first eventual action WQE is in field CNZMYQUAH_First_EA_Addr. The address of the first critical eventual action WQE is in field CNZMYQUAH_First_CEA_WQE_Addr. The WQE is mapped by IHAWQE and the address of the next WQE is in field WQELKP, with a zero value in WQELKP indicating that this is the last element of the queue. In addition, some status information about AMRF is returned (field CNZMYQUAH_AMRF_Status). Bit Cnzmyquah_Valid_AMRF_INFO is set to 1 when the information is successfully returned.

,ANSAREAALET=*ansareaalet*
A required input field that contains the ALET of the data space which is to contain the output information. The data space must be on the dispatchable unit access list or be a common area data space. It must include the address range X'1000' through X'7FFFEFFF' (that is, it is a 2G data space). It may contain the 0 and X'7FFFF000' pages. The area is mapped by macro CNZMYQUA. The header area, mapped by dsect CNZMYQUAHDR, will begin at location X'1000' in the data space.

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

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter in the "0–0" range that specifies the version of the macro. PLISTVER is the only key allowed on the list form of MF and determines which parakmeter list is generated. Note that MAX may be specified instead of a number, and the parameter list will be of the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list as that will ensure that the list form parameter list is always long enough to hold whatever parameters might be specified on the execute form.

DEFAULT: IMPLIED_VERSION. When PLISTVER is omitted, the default is the lowest version which allows all of the parameters specified on the invocation to be processed.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

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

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

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

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

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

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

ABEND codes

CNZQUERY might terminate abnormally with an abend code of X'0C2'. See *z/OS MVS System Codes* for an explanation and response for this code.

Return and reason codes

Return and reason code constants are defined in macro CNZMYQUA.

When the CNZQUERY macro returns control to your program, GPR 15 (and *retcode*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsncode*, if you coded RSNCODE) contains one of the following reason codes.

Table 28. Return and Reason Codes for the CNZQUERY Macro

Hexadecimal Return Code	Reason Code	Equate symbol Meaning and Action
00	00	**Equate symbol**: CNZQUERYRc_OK **Meaning**: CNZQUERY request successful. **Action**: None.
08		**Equate symbol**: CNZQUERYRc_InvParm **Meaning**: CNZQUERY request specifies invalid parameter. **Action**: Refer to action under the individual reason code.

Table 28. Return and Reason Codes for the CNZQUERY Macro (continued)

Hexadecimal Return Code	Reason Code	Equate symbol Meaning and Action
08	xxxx0801	**Equate symbol**: CnzqueryRsn_BadParmList **Meaning**: Unable to access parameter list. **Action**: Check for possible storage overlay.
08	xxxx0802	**Equate symbol**: CnzqueryRsn_SrbMode **Meaning**: SRB mode. **Action**: Avoid requesting this function in SRB mode.
08	xxxx0803	**Equate symbol**: CnzqueryRsn_NotEnabled **Meaning**: Not enabled. **Action**: Avoid requesting this function while not enabled.
08	xxxx0804	**Equate symbol**: CnzqueryRsn_BadAnsAreaALET **Meaning**: Bad answer area ALET. **Action**: Make sure that the ALET associated with the answer area is valid. The access register might not have been set up correctly.
08	xxxx0805	**Equate symbol**: CnzqueryRsn_BadAnsArea **Meaning**: Error accessing answer area. The data space might not have been defined to span 2G. **Action**: Make sure that the provided answer area is a valid 2G data space.
08	xxxx0806	**Equate symbol**: CnzqueryRsn_ReservedNot0 **Meaning**: Reserved field not 0. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0807	**Equate symbol**: CnzqueryRsn_BadParmlistALET **Meaning**: Bad parmlist ALET. **Action**: Make sure that the ALET of the parameter list is valid. The access register might not have been set up correctly.
08	xxxx0808	**Equate symbol**: CnzqueryRsn_BadVersion **Meaning**: Bad version number. **Action**: Check for possible storage overlay of the parameter list.
08	xxxx0809	**Equate symbol**: CnzqueryRsn_Locked **Meaning**: Locked. **Action**: Avoid requesting this function in this environment.
08	080A	**Equate symbol**: CnzqueryRsn_FRR **Meaning**: An FRR is set. **Action**: Avoid requesting this function in this environment.
10		**Equate symbol**: CnzqueryRC_CompError **Meaning**: Unexpected failure. **Action**: Refer to the action provided with the specific reason code.
10	xxxx1001	**Equate symbol**: CnzqueryRsn_CompError **Meaning**: Unexpected failure. The state of the request is unpredictable. **Action**: Contact your system programmer.

Example

```
*                                                                     * 25800000
*    Example 1                                                        * 25850000
*                                                                     * 25900000
*    Operation:                                                       * 25950000
*                                                                     * 26000000
*    After having established addressability and a dynamic area,      * 26050000
*                                                                     * 26100000
*                                                                     * 26150000
*    1.  Create a 2G data space                                       * 26200000
*                                                                     * 26250000
*    2.  Add the data space to the dispatchable unit access list      * 26300000
*                                                                     * 26350000
*    3.  Invoke CNZQUERY to retrieve WTOR and AMRF information        * 26400000
*                                                                     * 26450000
*    4.  Examine the WTOR queue                                       * 26500000
*                                                                     * 26550000
*    5.  Examine the AMRF Immediate Action queue                      * 26600000
*                                                                     * 26650000
*    6.  Delete the access list entry                                 * 26700000
*                                                                     * 26750000
*    7.  Delete the data space                                        * 26800000
*                                                                     * 26850000
*                                                                     * 26900000
*                                                                     * 26950000
*    The code is as follows.                                          * 27000000
*                                                                     * 27050000
*             SAC   512    Enter AR ASC mode                          * 27100000
*             SYSSTATE ASCENV=AR,ARCHLVL=2                            * 27150000
*    ************************************************************     * 27200000
*    * Create a 2G data space                                 *       * 27250000
*    ************************************************************     * 27300000
*             DSPSERV CREATE,NAME=dsName,BLOCKS=MaxBlocks,        *   * 27350000
*                   STOKEN=dsSTOKEN,                              *   * 27400000
*                   MF=(E,DSPSERVL)                                   * 27450000
*    *                                                                * 27500000
*    * Place code here to check return code from GPR 15 and           * 27550000
*    * reason code from GPR 0.                                        * 27600000
*    *                                                                * 27650000
*    ************************************************************     * 27700000
*    * Add the data space to the dispatchable unit access list   *    * 27750000
*    ************************************************************     * 27800000
*             ALESERV ADD,STOKEN=dsSTOKEN,ALET=dsALET,            *   * 27850000
*                   MF=(E,ALESERVL)                                   * 27900000
*    *                                                                * 27950000
*    * Place code here to check return code from GPR 15.              * 28000000
*    *                                                                * 28050000
*    ************************************************************     * 28100000
*    * Retrieve WTOR and AMRF information                     *       * 28150000
*    ************************************************************     * 28200000
*             CNZQUERY WTOR=YES,AMRF=YES,ANSAREAALET=dsALET,      *   * 28250000
*                   RETCODE=LRetcode,RSNCODE=LRsncode,            *   * 28300000
*                   MF=(E,CNZQUERYL)                                  * 28350000
*    *                                                                * 28400000
*    * Place code here to check return/reason codes.                  * 28450000
*    *                                                                * 28500000
*             LHI   2,HeaderAddr       Access header info            * 28550000
*             LAM   2,2,dsALET         With ALET                     * 28600000
*             USING CnzmyquaHdr,2                                     * 28650000
*    ************************************************************     * 28700000
*    * Examine the WTOR queue                                 *       * 28750000
*    ************************************************************     * 28800000
*             CPYA  3,2                ORE ALET = hdr ALET           * 28850000
*             ICM   3,B'1111',Cnzmyquah_First_ORE_Addr                * 28900000
*             JZ    NO_OREs                                           * 28950000
*             USING OREF,3                                            * 29000000
*             CPYA  4,2                WQE ALET = ORE ALET           * 29050000
*    NEXT_ORE DS    0H                                               * 29100000
*             L     4,ORERWQE                                         * 29150000
*             USING WQE,4                                             * 29200000
*    *                                                                * 29250000
*    * Place code here to examine the specific ORE and                * 29300000
*    * its associated WQE                                             * 29350000
*    *                                                                * 29400000
*             DROP  4                                                 * 29450000
*             ICM   3,B'1111',ORELKP                                  * 29500000
*             JNZ   NEXT_ORE                                          * 29550000
*             DROP  3                                                 * 29600000
*    NO_OREs  DS    0H                                               * 29650000
```

```
*    ************************************************************    * 29700000
*    * Examine the AMRF Immediate Action queue                  *    * 29750000
*    ************************************************************    * 29800000
*             CPYA  3,2                WQE ALET = hdr ALET           * 29850000
*             ICM   3,B'1111',Cnzmyquah_First_IA_WQE_Addr            * 29900000
*             JZ    NO_WQEs                                          * 29950000
*             USING WQE,3                                            * 30000000
*    NEXT_WQE DS    0H                                              * 30050000
*    *                                                              * 30100000
*    * Place code here to examine the specific WQE.                 * 30150000
*    *                                                              * 30200000
*             ICM   3,B'1111',WQELKP                                 * 30250000
*             JNZ   NEXT_WQE                                         * 30300000
*             DROP  3                                                * 30350000
*    NO_WQEs  DS    0H                                              * 30400000
*    ************************************************************    * 30450000
*    ************************************************************    * 30500000
*    * Delete the access list entry                             *    * 30550000
*    ************************************************************    * 30600000
*             ALESERV DELETE,ALET=dsALET,                        *   * 30650000
*                   MF=(E,ALESERVL)                                  * 30700000
*    *                                                              * 30750000
*    * Place code here to check return code from GPR 15.            * 30800000
*    *                                                              * 30850000
*    ************************************************************    * 30900000
*    * Delete the data space                                     *    * 30950000
*    ************************************************************    * 31000000
*             DSPSERV DELETE,STOKEN=dsSTOKEN,                    *   * 31050000
*                   MF=(E,DSPSERVL)                                  * 31100000
*    *                                                              * 31150000
*    * Place code here to check return code from GPR 15 and         * 31200000
*    * reason code from GPR 0.                                      * 31250000
*    *                                                              * 31300000
*    ************************************************************    * 31350000
*    * Exit the module                                           *    * 31400000
*    ************************************************************    * 31450000
*    *                                                              * 31500000
*    * Place code here to free the dynamic area and                 * 31550000
*    * exit the module.                                            * 31600000
*    * here                                                        * 31650000
*    *                                                              * 31700000
*    HeaderAddr EQU x'1000'     Where Cnzmyqua_Hdr is placed        * 31750000
*    STATAREA DS    0D                                              * 31800000
*    dsName   DC    CL8'MYDATASP'                                   * 31850000
*    MaxBlocks DC   A(524288)   Number of blocks in full 2G data space* 31900000
*    DYNAREA  DSECT                                                 * 31950000
*    dsSTOKEN DS    D                                               * 32000000
*    dsALET   DS    D                                               * 32050000
*    LRetcode DS    F                                               * 32100000
*    LRsncode DS    F                                               * 32150000
*    ListForms DS    0D                                             * 32200000
*             DSPSERV MF=(L,DSPSERVL)                               * 32250000
*             ORG   ListForms                                       * 32300000
*    ALESERVL ALESERV MF=L                                          * 32350000
*             ORG   ListForms                                       * 32400000
*             CNZQUERY MF=(L,CNZQUERYL)                             * 32450000
*             ORG   ,                                               * 32500000
*             CNZMYQUA              Output information              * 32550000
*             IHAWQE                WQE                             * 32600000
*             IHAORE                ORE                             * 32650000
*                                                                   * 32700000
```

Chapter 23. COFCREAT — Create a VLF object

Description

The COFCREAT macro allows your application to add an object, on behalf of an end user, to a class of VLF objects. Before issuing COFCREAT, or any VLF macro, you need to understand the information on using the virtual lookaside facility (VLF) that appears in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Normal processing of an end user request for an object includes the following steps:

1. Issue the COFRETRI macro to attempt to retrieve the object.
2. Examine the return code from COFRETRI. VLF can only create an object after you have tried to retrieve it and when COFRETRI completed with one of the following conditions:
   - Object not found (return code 8)
   - Best available object found (return code 2)
   - Best available object found, but target area is not large enough (return code 6)
3. If the return code is 8, create the object. (Processing return codes 2 or 6 might also require you to create the object.) Between issuing the COFRETRI and the COFCREAT for the object, do not issue any COFRETRI macro with the same UTOKEN.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary 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

- Your program must be running under a task with the same home ASID as the issuer of the COFIDENT macro that identified the user.
- For non-PDS classes, you can issue COFCREAT with the REPLACE option. If you specify REPLACE, VLF does not require that COFRETRI precede COFCREAT. Because VLF cannot then guarantee that the source object has not changed, your application must ensure that the source object remains unchanged between the time when you reference the source object to create the object parts list and the time when you receive control back from COFCREAT.

  If you do not specify REPLACE, you must issue the COFRETRI macro before you issue COFCREAT.
- To ensure the integrity of the data, the working storage that your application uses to create the VLF object must not be key 8 storage, and you must perform the following steps:

  1. Change to (or remain in) supervisor state.

2. Issue a BLDL macro for the PDS member using the same DDNAME used to identify the user to VLF. If a user changes the data set allocation associated with a DDNAME used to identify a VLF user, VLF invalidates that user's token (UTOKEN).
3. Save the “K” value from a successful BLDL to pass to VLF as the CINDEX value on COFCREAT.
4. Read the object from DASD, ensuring the following:
   - The DCB used for I/O must not be in key 8 storage.
   - The I/O buffers must not be in key 8 storage.
5. Issue the COFCREAT macro to create the VLF object.
6. If necessary, copy the object to key 8 storage to enable the user program to access it.

Failure to follow these steps compromises the integrity of data objects in VLF storage. Depending on the nature of the class of VLF objects, incorrect data could cause severe system integrity problems.

- If you do not specify REPLACE and issue COFCREAT for an object that already exists in VLF storage, VLF returns a successful completion code but does not replace the object data. In this case, VLF assumes that the data you supply is identical to the data that already exists in its storage.

  If you specify REPLACE and issue COFCREAT for an object that already exists in VLF storage, VLF does replace the existing object with the parts specified in the object parts list.

Restrictions

None.

Input register information

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

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-14
Unchanged

15
Return code

When control returns to the caller, the 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

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFCREAT
COFCREAT	
ƀ	One or more blanks must follow COFCREAT
MAJOR=*major*	*major*: Rx-type address or register (2) - (12).
CINDEX=*cindex*	*cindex*: Rx-type address or register (2) - (12).
,DDNAME=*ddname*	*ddname*: Rx-type address or register (2) - (12).
	Specify DDNAME only with CINDEX.
,REPLACE=NO	**Default:** REPLACE=NO
,REPLACE=YES	Specify REPLACE only with MAJOR.
,MINOR=*minor*	*minor*: Rx-type address or register (2) - (12).
,UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).
,OBJPRTL=*objprtl*	*objprtl*: Rx-type address or register (2) - (12).
,OBJPLSZ=*objplsz*	*objplsz*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).

Syntax	Description
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).

Parameters

The parameters are explained as follows:

MAJOR=*major*
Specifies the major name of the object to be created. The length of the major name must be the same as the length specified by MAJLEN on the COFDEFIN macro that defined the class of objects. Specify MAJOR only for a non-PDS class. (For a PDS class, you must use CINDEX and DDNAME.)

CINDEX=*cindex*
Identifies a one-byte field that contains the concatenation index of the major name associated with the object being created. CINDEX is required for a PDS class. The index is the zero-origin relative number of the major name for the object in the major name list of the user creating the object. This list is the one supplied to VLF on the COFIDENT macro that identified the user to VLF.

For concatenated partitioned data sets, the CINDEX value is the same as the “K” (concatenation index) value returned when your application issued a BLDL macro to locate a member.

When you specify CINDEX, you must also specify DDNAME.

,DDNAME=*ddname*
Specifies the 8-character DDNAME of the concatenated data set list. DDNAME is required for a PDS class. This DDNAME must be the same as the one supplied to VLF on the COFIDENT macro that identifies the user to VLF. It represents the major name search order for this identified user.

When you specify DDNAME, you must also specify CINDEX.

,REPLACE=YES
,REPLACE=NO
Indicates that an object existing in VLF should (REPLACE=YES) or should not (REPLACE=NO) be replaced by the parts in the input object parts list. If the object does not exist in VLF, then VLF creates a new object.

,MINOR=*minor*
Specifies the minor name of the object. The length of the significant portion of the name depends on the MINLEN value defined for the class on the COFDEFIN macro, either explicitly or by default. (For a PDS class, the length is always 8.)

,UTOKEN=*utoken*
Specifies the required 16-character user token returned from the COFIDENT macro for the user on whose behalf your application is issuing COFCREAT.

,OBJPRTL=*objprtl*
Specifies the required object parts list. The object parts list describes the source areas from which VLF can obtain consecutive parts of the object. The object parts list consists of a fullword containing the number of object parts, followed by three words for each part:

1. A fullword that contains the ALET that currently addresses the part. An ALET of 1, referencing the SASN of the caller, or ALETs referencing entries on the PASN access list of the caller, are not allowed.
2. A fullword that contains the 31-bit address of the data for the part.
3. A fullword that contains the length of the part.

The number of parts list entries must be from 1 to 16. If your program is not running in access register (AR) ASC mode, the ALET(s) must be zero.

,OBJPLSZ=*objplsz*
Specifies the required fullword field that contains the size (in bytes) of the object parts list.

,RETCODE=*retcod*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary,

,RSNCODE=*rsncod*
Specifies the location where the system is to store the reason code. If you specify a storage location, it must be on a fullword boundary. The reason code is also in GPR 0.

ABEND codes

None.

Return and reason codes

When the COFCREAT macro returns control to your program, GPR 15 (and *retcod*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsncod*, if you coded RSNCODE) contains one of the following reason codes.

Table 29. Return and Reason Codes for the COFCREAT Macro

Hexadecimal Return Code	Reason Code	Meaning and Action
00	00	**Meaning**: The VLF object has been created. **Action**: None.
02	02	**Meaning**: Program error. No VLF object was created because the create request specified an ineligible major name. **Action**: Check the major name specified on the macro invocation. Ensure that there is a matching EMAJ specified major name for this class in the SYS1.PARMLIB member COFVLFnn.
02	04	**Meaning**: Environmental error. No VLF object was created. A retrieve request was not done for this minor name, a time-out occurred for the pending create, or the pending create was invalidated by a notification that the object might have changed. **Action**: Issue the COFRETRI macro prior to issuing the COFCREAT macro. It may be necessary to retry the COFRETRI/COFCREAT invocations several times.
02	06	**Meaning**: Environmental error. No VLF object was created because the create request specified a major name that has been invalidated by a notification that the object might have changed. The user token will be invalidated. This situation also can occur as a result of a Modify VLF command that changes the major name. **Action**: Issue a COFIDENT macro to reidentify the user to VLF.
04	00	**Meaning**: Program error. The requested major name is not in the user's search order. **Action**: Ensure that the requested major name was specified in the search order specified through the MAJNLST when the user was identified with the COFIDENT macro.
0A	00	**Meaning**: Program error. The parameter list cannot be accessed. **Action**: Make necessary corrections to ensure that the parameter list ALET is on the dispatchable unit access list (DU-AL) and rerun the job.
0C	00	**Meaning**: Program error. The class to which the user is identified is not currently defined. **Action**: Redefine the class with COFDEFIN and retry the COFCREAT.
10	00	**Meaning**: Program error. A user token was specified but the user is not currently identified to VLF. **Action**: Identify the user with COFIDENT and retry the COFCREAT.

Table 29. Return and Reason Codes for the COFCREAT Macro (continued)

Hexadecimal Return Code	Reason Code	Meaning and Action
12	00	**Meaning**: Program error. The DDNAME is not the same as the DDNAME specified on the COFIDENT macro that returned this user token. **Action**: Use the same DDNAME that was specified with the COFIDENT, and retry the COFCREAT.
14	00	**Meaning**: Environmental error. VLF incurred a program check when it tried to access the object parts list. **Action**: Retry the operation. If the problem persists, specify a smaller OBJPLSZ parameter for the OBJPRTL.
18	00	**Meaning**: Program error. The class to which the user is identified is a PDS class, but CINDEX was not specified. **Action**: Reissue the COFCREAT, and include the required CINDEX keyword.
18	02	**Meaning**: Program error. OBJPLSZ was larger than the maximum allowable size, or the number of parts in the object parts list was greater than 16. **Action**: Ensure that the specified OBJPLSZ was not greater than 16, and that the number of object parts specified in OBJPE4RTL is not greater than 16, and then reissue the COFCREAT macro.
18	04	**Meaning**: Program error. REPLACE was specified, but the class to which the user is identified is a PDS class. **Action**: Reissue the COFCREAT without specifying the REPLACE option.
18	0A	**Meaning**: Program error. The major name cannot be accessed by the specified ALET. The ALET is a SASN ALET, or the ALET is not on the dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the major name ALET parameter is on the dispatchable unit access list (DU-AL) and rerun the job.
18	0B	**Meaning**: Program error. The minor name cannot be accessed by the specified ALET. The ALET is a SASN ALET, or the ALET is not on the dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the minor name ALET parameter is on the dispatchable unit access list (DU-AL) and rerun the job.
18	0C	**Meaning**: Program error. The object parts list cannot be accessed using the specified ALET. The ALET is a SASN ALET, or the ALET is not on the dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the objects parts list ALET parameter is on the dispatchable unit access list (DU-AL) and rerun the job.
18	0D	**Meaning**: Program error. A part in the object parts list cannot be accessed using the specified ALET. The ALET is a SASN ALET, or the ALET is not on the dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the objects parts list ALET parameter is on the dispatchable unit access list (DU-AL) and rerun the job.
1C	00	**Meaning**: Environmental error. There was not enough storage available to create this object. **Action**: Increase the value of MAXVIRT for this class in the SYS1.PARMLIB member COFVLFnn; or ensure that TRIM=ON is specified when the class is defined with COFDEFIN; or free storage through the use of COFPURGE.
28	00	**Meaning**: Environmental error. VLF is not active. **Action**: Issue the Start VLF system command, and rerun the job.
2C	nnnn	**Meaning**: System error. There was an unexpected error in VLF. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

COFCREAT - List form

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

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

Parameters

The parameters of the list form are explained as follows:

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

COFCREAT - Execute form

Syntax

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

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

Syntax	Description
ƀ	One or more blanks must precede COFCREAT
COFCREAT	
ƀ	One or more blanks must follow COFCREAT
MAJOR*major*	*major*: Rx-type address or register (2) - (12).
,CINDEX=*cindex*	*cindex*: Rx-type address or register (2) - (12).
,DDNAME=*ddname*	*ddname*: Rx-type address or register (2) - (12).
	Specify DDNAME only with CINDEX.
,REPLACE=YES	Specify REPLACE only with MAJOR.
,REPLACE=NO	**Default:** REPLACE=NO
,MINOR=*minor*	*retcod*: Rx-type address or register (2) - (12).
,UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).
,OBJPRTL=*objprtl*	*objprtl*: Rx-type address or register (2) - (12).
,OBJPLSZ=*objplsz*	*objplsz*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*retcod*: Rx-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: Rx-type address or register (2) - (12).

Parameters

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

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

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

Chapter 24. COFDEFIN — Define a VLF class

Description

COFDEFIN defines a class of virtual lookaside facility (VLF) objects. Before issuing COFDEFIN, or any VLF macro, you need to understand the information on using the VLF that appears in *z/OS MVS Programming: Authorized Assembler Services Guide*.

When you define a class of VLF objects, the system allocates virtual storage for the class and generates the necessary control blocks. If the class has already been defined, VLF rejects the request. The maximum amount of virtual storage available for the class can be controlled by the MAXVIRT keyword on the CLASS statement in the COFVLFxx parmlib member. When the MAXVIRT keyword is not used, the default is 4096 pages.

The system obtains the attributes of the class from the input parameters of the macro and the description of the class in the active COFVLFxx parmlib member.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Upon invocation, the general purpose registers (GPRs) must contain:

Register
Contents

1
Address of parameter list

13
Address of standard 72-byte 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

2-14
Unchanged

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFDEFIN
COFDEFIN	
ƀ	One or more blanks must follow COFDEFIN
CLASS=*class*	*class*: RX-type address or register (2) - (12).
,MAJLEN=*majlen*	*majlen*: RX-type address or register (2) - (12).

Syntax	Description
,MINLEN=*minlen*	*majlen*: RX-type address or register (2) - (12).
,TRIM=ON	**Default**: ON
,TRIM=OFF	
,AUTHRET=NO	**Default**: NO
,AUTHRET=YES	
,RETCODE=*retcod*	*retcod*: RX-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: RX-type address or register (2) - (12).

Parameters

The parameters of the standard form are as follows:

CLASS=*class*
Specifies a 7-byte field that identifies the name of the class of VLF objects to be created. The name, which can be from 1 to 7 characters, can consist of any combination of upper case alphabetic and numeric characters and @, #, and $. The name must match the name of a class described in the active COFVLFxx parmlib member.

IBM-supplied VLF class names begin with the uppercase letters A-I. Choose names for installation-supplied VLF classes that begin with J-Z, numeric characters, or @, #, or $.

,MAJLEN=*majlen*
Identifies a 1-byte field specifying the length, from 1 to 64 bytes, of the major names in this class. This parameter is required for a non-PDS class. For a PDS class, the length is always 50.

,MINLEN=*minlen*
Identifies a 1-byte field specifying the length, from 1 to 64 bytes, of the minor names in this class. This parameter is required for a non-PDS class. For a PDS class, the length is always 8.

,TRIM=ON
,TRIM=OFF
An optional parameter that specifies how you want VLF to manage virtual storage for the objects in the class. If you specify TRIM=ON, which is the default, VLF automatically removes the least recently used objects when it needs space. If you specify TRIM=OFF, VLF removes objects only when it is specifically notified. Allowing VLF to manage the storage (TRIM=ON) ensures that, if space is limited, the most recently used objects tend to remain in virtual storage.

,AUTHRET=NO
,AUTHRET=YES
An optional parameter that indicates whether tasks that issue the COFRETRI macro to retrieve objects from the class must be in supervisor state or have PSW key mask 0-7. To restrict retrieves for the class to such tasks, specify AUTHRET=YES. The default is AUTHRET=NO.

,RETCODE=*retcod*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsncod*
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 the COFDEFIN macro returns control to your program, GPR 15 (and *retcod*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsncod*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Table 30. Return and Reason Codes for the COFDEFIN Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: The define request was successful. **Action**: None.
02	04	**Meaning**: A define request for the same class is currently in progress. **Action**: None required.
02	08	**Meaning**: The class is already defined. **Action**: You must issue COFPURGE for the class before you can redefine the class.
02	0C	**Meaning**: The class is already defined. VLF has changed the existing class definition to require that issuers of COFRETRI for the class be in supervisor state or have PSW key mask 0-7. **Action**: You must issue COFPURGE for the class before you can redefine the class.
04	00	**Meaning**: Environmental error. The define request failed. The class state is not valid. **Action**: Rerun the program.
08	00	**Meaning**: Environmental error. A purge request for the same class was issued before the define request completed. **Action**: Rerun the program.
08	04	**Meaning**: Environmental error. The class was being purged when you issued COFDEFIN. **Action**: Rerun the program.
0C	00	**Meaning**: Program error. There was no description for the class in the active COFVLFxx parmlib member. **Action**: Check that the class specified in the macro invocation matches a class specified in the SYS1.PARMLIB member COFVLFxx.
10	04	**Meaning**: Program error. The value for MAJLEN is not within the allowed range. **Action**: Specify a value between 1 and 64 for MAJLEN, and rerun the program.
10	08	**Meaning**: Program error. The value for MINLEN is not within the allowed range. **Action**: Specify a value between 1 and 64 for MINLEN, and rerun the program.

Table 30. Return and Reason Codes for the COFDEFIN Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
10	0C	**Meaning**: Program error. The values for both MAJLEN and MINLEN are not within the allowed range. **Action**: Specify a value between 1 and 64 for both MAJLEN and MINLEN, and rerun the program.
18	00	**Meaning**: Program error. The parameter list ALET is not valid. **Action**: Make necessary corrections in the application, and rerun the job.
28	00	**Meaning**: Environmental error. VLF is not active. **Action**: Issue the Start VLF command, and rerun the job.
2C	nnnn	**Meaning**: System error. There was an unexpected error in VLF. **Action**: Record the return and reason code and supply it to the appropriate IBM support personnel.

COFDEFIN - List form

Syntax

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

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

Parameters

The parameters of the list form are as follows:

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

COFDEFIN - Execute form

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFDEFIN
COFDEFIN	
ƀ	One or more blanks must follow COFDEFIN
CLASS=*class*	*class*: RX-type address or register (2) - (12).
,MAJLEN=*majlen*	*majlen*: RX-type address or register (2) - (12).
,MINLEN=*minlen*	*majlen*: RX-type address or register (2) - (12).
,TRIM=ON	**Default**: ON
,TRIM=OFF	
,AUTHRET=YES	**Default**: NO
,AUTHRET=NO	
,RETCODE=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).

Parameters

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

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

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

Chapter 25. COFIDENT — Identify a VLF user

Description

The COFIDENT macro allows an individual user to access a particular class of VLF objects. Before issuing COFIDENT, or any VLF macro, you need to understand the information on using the virtual lookaside facility (VLF) that appears in *z/OS MVS Programming: Authorized Assembler Services Guide*.

You must issue COFIDENT to identify the class and user before VLF can retrieve or create objects on behalf of that user. With COFIDENT, you also specify to VLF the search order it is to use to locate objects for the user.

As part of COFIDENT processing, VLF returns a unique user token (UTOKEN). The user token identifies the user (through an associated home ASID), class, and search order. Other VLF functions, such as retrieving or creating objects, require you to supply this user token.

The value of the user token returned by the successful completion of this function is never zero. Thus, you can check a saved user token field for zero to determine if an end user has been identified to VLF.

If the end user has private data sets in a DDNAME concatenation (data sets not defined for this class in the active COFVLFxx parmlib member), they are not eligible data sets. That is, VLF does not use them as a source of VLF objects.

If you have control over the search orders, VLF works most efficiently when private data sets (or ineligible major names for non-PDS classes) are either not allowed or follow the eligible names rather than precede them.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with PSW key 0-7
Dispatchable unit mode:	Task for PDS class (if you specify DDNAME); task or SRB for non-PDS class (if you specify MAJNLST)
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

- Before obtaining the user token, you must ensure that the user is authorized to access the objects. Open the DDNAME or perform authority checking before you issue the COFIDENT macro.
- The storage area to be used for the parameter list must reside in the caller's primary address space. The ALET used to qualify this storage must be 0.
- When you specify DDNAME, you must issue the COFIDENT macro from a task running under the same home ASID as the task that allocated the DDNAME.

- When you specify SCOPE=HOME or take the default, the returned user token (UTOKEN) is valid only for tasks with the same home ASID as the issuer of the COFIDENT macro. Subsequent VLF macros (COFCREAT, COFRETRI, or COFREMOV) that supply this user token must have the same home ASID.
- When you specify SCOPE=SYSTEM, the issuers of the COFCREAT and COFREMOV macros must have the same home ASID as the issuer of COFIDENT. However, the COFRETRI macro can be issued by tasks that have a home ASID that is different from the home ASID of the issuer of the COFIDENT macro. VLF treats a COFRETRI macro issued with this UTOKEN as if the request had come from the task that issued the COFIDENT macro. Any task that supplies the UTOKEN can retrieve objects created with the UTOKEN unless the COFDEFIN macro that defined the class specified AUTHRET=YES. In this case, only supervisor-state tasks, or tasks running with PSW key 0-7, can retrieve objects from the class.

Restrictions

None.

Input register information

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

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code

1
Used as a work register by the system

2-14
Unchanged

15
Return code

When control returns to the caller, the 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

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFIDENT
COFIDENT	
ƀ	One or more blanks must follow COFIDENT
DDNAME=*ddname*	*ddname*: Rx-type address or register (2) - (12).
MAJNLST=*majnlst*	*majnlst*: Rx-type address or register (2) - (12).
,CLASS=*class*	*class*: Rx-type address or register (2) - (12).
,SCOPE=HOME	**Default**: HOME
,SCOPE=SYSTEM	
,UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).

Parameters

The parameters of the standard form are explained as follows:

DDNAME=*ddname*
Specifies, for a PDS class, the ddname of a concatenated data set list. When VLF locates objects on behalf of this user, it uses the order in which data sets appear in this data set list as its search order. Note that the concatenated data set list can contain private data sets; VLF creates objects, however, only from eligible data sets (data sets included in the class description in the active COFVLFxx parmlib member). Specify DDNAME only for PDS classes.

Note: Before you issue COFIDENT, you must verify that the end user is authorized to access any data sets referenced by this DDNAME. Open the DDNAME before issuing the COFIDENT macro to ensure that the end user has authority to access the data sets in the DDNAME concatenation.

If you specify DDNAME, do not specify MAJNLIST.

MAJNLST=*majnlst*
Defines, for non-PDS classes, the search order VLF is to use to locate objects for this user. Each entry in the list must match a major name defined for the class through EMAJ in the active COFVLFxx parmlib member.

MAJNLST is required for a non-PDS class. The list that *majnlst* points to consists of a 4-byte field containing the number of entries in the list, followed by a contiguous list of from 1 to 256 major names. The list must contain at least one entry.

Each name in the list must be the same length, padded with blanks on the right if necessary. The length of each name in the list must be equal to the length supplied for MAJLEN on the COFDEFIN macro when the class was defined.

Note that the variable name of the major name list may be ALET qualified, but that an ALET of 1, referencing the SASN of the caller, or ALETs referencing entries on the PASN access list of the caller, are not allowed.

If you specify MAJNLST, do not specify DDNAME.

,CLASS=*class*
Specifies the required 7-character name of a VLF class, already defined to VLF through the COFDEFIN macro.

,SCOPE=HOME
,SCOPE=SYSTEM
An optional parameter that indicates the scope of services that can retrieve objects with the UTOKEN returned by this COFIDENT. The default is HOME.

HOME indicates that only services with the same home ASID as the task issuing the COFIDENT macro can retrieve objects with the returned user token (UTOKEN).

SYSTEM indicates that services with a home ASID different from that of the task issuing the COFIDENT macro can retrieve objects with the returned user token (UTOKEN). In this case, a COFRETRI macro issued with this UTOKEN is treated as if the request had come from the task that issued the COFIDENT macro. SCOPE=SYSTEM allows a service running under a particular home ASID to control a set of VLF objects and allow all tasks in the system to access those objects.

,UTOKEN=*utoken*
Specifies a required 16-character output variable that contains the unique user token value that VLF returns to identify this user. You will provide this value on subsequent requests to create or retrieve VLF objects on behalf of this user.

,RETCODE=*retcod*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsncod*
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 the COFIDENT macro returns control to your program, GPR 15 (and *retcod*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsncod*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Table 31. Return and Reason Codes for the COFIDENT Macro

Hexadecimal Return Code	Reason Code	Meaning and Action
00	00	**Meaning**: Successful completion. The user has been identified to VLF with the specified major name search order. **Action**: None.

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

Hexadecimal Return Code	Reason Code	Meaning and Action
02	08	**Meaning**: The user is already identified to VLF for this class. The user token from the previous IDENTIFY has been returned in the UTOKEN field. **Action**: None required.
04	00	**Meaning**: Environmental error. The identify request cannot be completed. Another identify request from the same home ASID is currently in progress for the same class and DDNAME. **Action**: Rerun the program.
08	00	**Meaning**: Program error. No major names in the search order contain objects that are eligible objects for VLF. The data sets listed in the search order will not be cached by VLF. **Action**: None required.
0C	00	**Meaning**: Environmental error. The class has not been defined to VLF. **Action**: Issue COFDEFIN for this class and retry the COFIDENT.
10	00	**Meaning**: Program error. VLF could not obtain the list of partitioned data sets for the input DDNAME. The task invoking VLF might not have been running under the same home ASID as the task that allocated the DDNAME. **Action**: Issue the COFREMOV macro to remove the user from VLF. Then issue COFIDENT to reidentify the user, and rerun the program.
14	00	**Meaning**: Program error. There was an incorrect input parameter. Either the DDNAME keyword was not specified for an input PDS class, or the DDNAME keyword was specified for a non-PDS class. **Action**: If the class specified is a PDS, you must also specify the DDNAME keyword. If the class specified is a non-PDS, you must not specify the DDNAME keyword. Make the appropriate correction and rerun the program.
18	08	**Meaning**: Program error. The number of major names in a search order is not in the range 1 through 256. **Action**: The first word in the list pointed to by MAJNLST must contain a number from 1 through 256. Make this correction and retry the COFIDENT.
18	0C	**Meaning**: Program error. The input major name list was qualified using either a SASN ALET or an ALET not on the caller's dispatchable unit access list (DU-AL). **Action**: Make the necessary corrections to ensure that the input major name list ALET is on the dispatchable unit access list (DU-AL) and rerun the job.
1C	04	**Meaning**: Program error. The DDNAME was not open. **Action**: Open the DDNAME and rerun the program.
1C	08	**Meaning**: Program error. The DDNAME was not allocated. **Action**: Allocate the DDNAME before you issue the COFIDENT macro.
1C	12	**Meaning**: Environmental error. The DDNAME concatenation was changed without deallocating the DDNAME. VLF no longer accepts user identification requests that specify the DDNAME. **Action**: Issue the COFREMOV macro to remove the user from VLF. Then issue COFIDENT to reidentify the user.
28	00	**Meaning**: Environmental error. VLF is not active. **Action**: Issue the START VLF command, and rerun the job.
2C	nnnn	**Meaning**: System error. There was an unexpected error in VLF. nnnn is the reason code. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

COFIDENT - List form

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

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

Parameters

The parameters of the list form are explained as follows:

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

COFIDENT - Execute form

Syntax

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

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

Syntax	Description
ƀ	One or more blanks must follow COFIDENT
DDNAME=*ddname*	*ddname*: Rx-type address or register (2) - (12).
MAJNLST=*majnlst*	*majnlst*: Rx-type address or register (2) - (12).
,CLASS=*class*	*class*: Rx-type address or register (2) - (12).
,SCOPE=HONE	**Default**: HOME
,SCOPE=SYSTEM	
,UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: Rx-type address or register (2) - (12).

Parameters

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

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

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

Chapter 26. COFNOTIF — Notify VLF

Description

The COFNOTIF macro allows an application using VLF to notify VLF that some set of VLF objects is no longer valid because of changes to the permanent data. Before issuing COFNOTIF, or any VLF macro, you need to understand the information on using the virtual lookaside facility (VLF) that appears in *z/OS MVS Programming: Authorized Assembler Services Guide*.

You can issue COFNOTIF to notify VLF about the following kinds of changes:

- One or more major names have been deleted. You must specify FUNC=DELMAJOR and MAJLIST.

  You might need to specify MAJNUM and MAJLEN, and you also might need to specify CLASS.
- One or more minor names have been changed. You must specify FUNC=DELMINOR (for a deletion), FUNC=ADDMINOR (for an addition), or FUNC=UPDMINOR (for a change). You must also specify MAJOR and MINLIST.

  You might need to specify MINNUM and MINLEN, and you also might need to specify CLASS.
- A volume is no longer in use. You must specify FUNC=PURGEVOL and VOLUME.

Note that an update to a minor name with one or more alias names means that you must specify the minor name and each alias name. VLF views each alias name as a separate minor name and thus needs to know about the update under each name.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the COFNOTIF 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

2-14
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

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFNOTIF
COFNOTIF	
ƀ	One or more blanks must follow COFNOTIF
FUNC=DELMAJOR	
FUNC=DELMINOR	
FUNC=ADDMINOR	

Syntax	Description
FUNC=UPDMINOR	
FUNC=PURGEVOL	
,MAJLIST=*majlist*	*majlist*: Rx-type address or register (2) - (12).
	You must specify MAJLIST=*majlist* when you specify FUNC=DELMAJOR.
,MAJNUM=*majnum*	*majnum*: Rx-type address or register (2) - (12).
,MAJLEN=*majlen*	*majlen*: Rx-type address or register (2) - (12).
,MAJOR=*major*	*major.* Rx-type address or register (2) - (12).
	You must specify MAJLIST=*major* when you specify FUNC=DELMINOR, FUNC=ADDMINOR, or FUNC=UPDMINOR.
,MINLIST=*minlist*	*minlist*. Rx-type address or register (2) - (12).
	You must specify MINLIST=*minlist* when you specify FUNC=DELMINOR, FUNC=ADDMINOR, or FUNC=UPDMINOR.
,MINNUM=*minnum*	*minnum*: Rx-type address or register (2) - (12).
,MINLEN=*minlen*	*minlen*: Rx-type address or register (2) - (12).
,VOLUME=*volume*	*volume*: Rx-type address or register (2) - (12).
,CLASS=*class*	*class*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).

Parameters

The parameters of the standard form are explained as follows:

FUNC=DELMAJOR
FUNC=DELMINOR
FUNC=ADDMINOR
FUNC=UPDMINOR
FUNC=PURGEVOL

Is a required parameter that indicates the nature of the change that you are reporting. The meaning of each value is as follows:

- FUNC=DELMAJOR specifies that one or more major names have been deleted.
- FUNC=DELMINOR specifies that one or more minor names have been deleted from a major name.
- FUNC=ADDMINOR specifies that one or more minor names have been added to a major name.
- FUNC=UPDMINOR specifies that the objects corresponding to one or more existing minor names have been changed.
- FUNC=PURGEVOL specifies that a physical storage device has been logically disconnected from the system, or that all of the information on the device has been deleted or replaced.

,MAJLIST=*majlist*

Identifies the list of major names with which the change is associated. When you specify FUNC=DELMAJOR, you must specify MAJLIST to identify the major name(s) VLF is to delete. If the list contains more than one major name, you must also specify MAJNUM. Each major name in the list must be the same length. If the major name length is not 64, you must also specify MAJLEN.

Use the following structure to specify the major name for a PDS class:

- 6-character volume serial name (padded with blanks if necessary)
- PDS name (a maximum of 44 characters), padded with blanks to equal 64 or the MAJLEN value.

For example, assume that you want to delete the major name MYPDS that resides on volume VOL123. Specify VOL123MYPDS, padded with blanks as required.

,MAJNUM=*majnum*

An optional halfword parameter that contains the number of major names in the major name list. The default is 1.

,MAJLEN=*majlen*

An optional halfword parameter that contains the length of each input major name. The default is 64.

Note: VLF uses the length you specify to scan the major name list. The length of the significant part of the name (the part VLF uses to search its storage for objects with that major name) depends on the value specified for the major name on the COFDEFIN macro that defined the class. If the COFDEFIN length is greater than the COFNOTIF length, VLF pads the name on the right with blanks.

,MAJOR=*major*

Identifies the major name associated with the change to one or more minor names. When you specify FUNC=DELMINOR, FUNC=ADDMINOR, or FUNC=DELMINOR, you must specify MAJOR. If the length is not 64, you must also specify MAJLEN.

Use the following structure to specify the major name for a PDS class:

- 6-character volume serial name (padded with blanks if necessary)
- PDS name (a maximum of 44 characters), padded with blanks to equal 64 or the MAJLEN value.

For example, assume that you want to delete the major name MYPDS that resides on volume VOL123. Specify VOL123MYPDS, padded with blanks as required.

,MINLIST=*minlist*

Identifies the list of minor names with which the change is associated. When you specify FUNC=DELMINOR, FUNC=ADDMINOR, or FUNC=UPDMINOR, you must specify MINLIST. If the list contains more than one minor name, you must also specify MINNUM. If the length is not 64, then you must also specify MINLEN. Each name in the list must be the same length.

,MINNUM=*minnum*
An optional halfword parameter that contains the number of minor names in the minor name list. The default is 1.

,MINLEN=*minlen*
An optional halfword parameter that contains the length of each name in the input minor name list. The default is 64.

Note: VLF uses the length you specify to scan the minor name list. The length of the significant part of the name (the part VLF uses to search its storage for objects with that minor name) depends on the value specified for the minor name on the COFDEFIN macro that defined the class. If the COFDEFIN length is greater than the COFNOTIF length, VLF pads the name on the right with blanks.

,VOLUME=*volume*
Specifies the volume serial number of a resource that was logically removed from the system. Specifying VOLUME causes VLF to purge any objects related to the resource identified.

Specify VOLUME only for objects with major names that correspond to PDS names and only when you also specify FUNC=PURGEVOL.

,CLASS=*class*
Specifies a 7-byte field that identifies the name of the class associated with the change. CLASS is an optional parameter. Specify CLASS only for a non-PDS class. If you omit CLASS or specify a PDS class, VLF assumes that the change being reported applies to all PDS classes.

,RETCODE=*retcod*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsncod*
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 the COFNOTIF macro returns control to your program, GPR 15 (and *retcod*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsncod*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Table 32. Return and Reason Codes for the COFNOTIF Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: Successful completion. VLF now reflects the indicated changes. **Action**: None.
02	08	**Meaning**: No changes to VLF storage were necessary. **Action**: None.
02	0C	**Meaning**: The specified class was not defined to VLF. This code is returned only for an input class that does not have a major name to PDS correspondence. No changes to VLF storage occurred. **Action**: None.
02	10	**Meaning**: The specified class is not defined in the active COFVLFxx parmlib member. No changes to VLF storage occurred. **Action**: None required.

Table 32. Return and Reason Codes for the COFNOTIF Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
18	00	**Meaning**: Program error. The parameter list ALET is either a SASN ALET or is not on the caller's dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the parameter list ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
18	08	**Meaning**: Program error. The input major name was qualified using either a SASN ALET or an ALET not on the caller's dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the major name ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
18	0C	**Meaning**: Program error. The input minor name was qualified using either a SASN ALET or an ALET not on the caller's dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the minor name ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
1C	nnnn	**Meaning**: Program error. An error occurred while accessing a major name in the input major name list; *nnnn* identifies the list position of the major name that caused the error. COFNOTIF processing terminates. **Action**: Check parameters such as MAJNUM and MAJLEN for accuracy. Make necessary corrections and rerun the program.
20	nnnn	**Meaning**: Program error. An error occurred while accessing a minor name in the input minor name list; *nnnn* identifies the list position of the minor name that caused the error. COFNOTIF processing terminates. **Action**: Check parameters such as MINNUM and MINLEN for accuracy. Make necessary corrections and rerun the program.
28	00	**Meaning**: Environmental error. VLF is not active. **Action**: Issue the START VLF command and rerun the program.
2C	nnnn	**Meaning**: System error. There was an unexpected error in VLF. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

COFNOTIF - List form

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

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

Syntax	Description
ƀ	One or more blanks must follow COFNOTIF
MF=(L,*mfctrl*)	*mfctrl*: Symbol.
MF=(L,*mfctrl*,*mfattr*)	*mfattr*: 1- to 60-character input string. **Default:** 0D.

Parameters

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

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

list addr is the name of a storage area to contain the parameters. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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.

COFNOTIF - Execute form

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFNOTIF
COFNOTIF	
ƀ	One or more blanks must follow COFNOTIF
FUNC=DELMAJOR	
FUNC=DELMINOR	
FUNC=ADDMINOR	
FUNC=UPDMINOR	

Syntax	Description
FUNC=PURGEVOL	
,MAJLIST=*majlist*	*majlist*: RX-type address or register (2) - (12).
	You must specify MAJLIST=*majlist* when you specify FUNC=DELMAJOR.
,MAJNUM=*majnum*	*majnum*: RX-type address or register (2) - (12).
,MAJLEN=*majlen*	*majlen*: RX-type address or register (2) - (12).
,MAJOR=*major*	*major*. RX-type address or register (2) - (12).
	You must specify MAJOR=*major* when you specify FUNC=DELMINOR, FUNC=ADDMINOR, or FUNC=UPDMINOR.
,MINLIST=*minlist*	*minlist*. RX-type address or register (2) - (12).
	You must specify MINLIST=*minlist* when you specify FUNC=DELMINOR, FUNC=ADDMINOR, or FUNC=UPDMINOR.
,MINNUM=*minnum*	*minnum*: FIXED(15) field or register (2) - (12).
,MINLEN=*minlen*	*minlen*: FIXED(15) field or register (2) - (12).
,VOLUME=*volume*	*volume*: Rx-type address or register (2) - (12).
,CLASS=*class*	*class*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: Rx-type address or register (2) - (12).

Parameters

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

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

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

Chapter 27. COFPURGE — Purge a VLF class

Description

The COFPURGE macro requests that VLF purge (delete) a class of VLF objects. Before issuing COFPURGE, or any VLF macro, you need to understand the information on using the virtual lookaside facility (VLF) that appears in *z/OS MVS Programming: Authorized Assembler Services Guide*.

When you issue COFPURGE, VLF deletes the class immediately. Any transaction in process for the purged class fails; VLF issues a failure return code that is appropriate for the transaction. To reinstate the class, you must issue another COFDEFIN for the class, which you can do at any time. Once you have reinstated the class, you must identify the users of the class again.

Note that the system can also delete a class for control purposes even if no user requests it. Your application learns that the system has purged a class when it issues a COFIDENT, COFREMOV, COFCREAT, or COFRETRI macro specifying that class. There are specific return and reason code combinations to distinguish a class that is not defined from other error indicators.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the COFPURGE 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

2-14
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

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFPURGE
COFPURGE	
ƀ	One or more blanks must follow COFPURGE
CLASS=*class*	*class*: RX-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: RX-type address or register (2) - (12).
,RSNCODE=*rsncod*	*retcod*: RX-type address or register (2) - (12).

Parameters

The parameters of the standard form are as follows:

CLASS=*class*
Specifies the required name of the class of VLF objects to be deleted.

,RETCODE=*retcod*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsncod*
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 the COFPURGE macro returns control to your program, GPR 15 (and *retcod*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsncod*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Table 33. Return and Reason Codes for the COFPURGE Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: The purge was successful. The class is no longer described to VLF. **Action**: None.
02	04	**Meaning**: Program error. The specified class was not described in the active COFVLFxx parmlib member. **Action**: Check that the class specified in the macro invocation matches a class specified in the SYS1.PARMLIB member COFVLFxx.
28	00	**Meaning**: Environmental error. VLF is not active. **Action**: Issue the START VLF command and rerun the program.

COFPURGE - List form

Syntax

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

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

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

Parameters

The parameters of the list form are as follows:

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

list addr is the name of a storage area to contain the parameters. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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.

COFPURGE - Execute form

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFPURGE
COFPURGE	
ƀ	One or more blanks must follow COFPURGE
CLASS=*class*	*class*: RX-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).

Parameters

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

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

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

Chapter 28. COFREMOV — Remove a VLF user

Description

COFREMOV terminates an end user's access to the class of VLF objects associated with the specified user token (UTOKEN). Before issuing COFREMOV, or any VLF macro, you need to understand the information on using the virtual lookaside facility (VLF) that appears in *z/OS MVS Programming: Authorized Assembler Services Guide*.

You issue COFREMOV when your program determines that an end user should no longer have access to the class of VLF objects. You must supply the same user token (UTOKEN) on COFREMOV that VLF returned on the COFIDENT macro that identified the user. You must issue COFREMOV from a task that has the same home ASID as the task that issued the COFIDENT to identify the user.

After you have removed the user, VLF rejects, with a reason code that indicates an unknown UTOKEN, any subsequent VLF requests that specify the UTOKEN.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

None.

Input register information

Before issuing the COFREMOV 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

2-14
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

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFREMOV
COFREMOV	
ƀ	One or more blanks must follow COFREMOV
UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).

Parameters

The parameters of the standard form are explained as follows:

UTOKEN=*utoken*
Specifies a required 16-character input parameter that contains the user token value (obtained from the COFIDENT macro) for the user you are removing from VLF.

,RETCODE=*retcod*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsncod*
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 the COFREMOV macro returns control to your program, GPR 15 (and *retcod*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsncod*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Table 34. Return and Reason Codes for the COFREMOV Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: Successful completion. The record of the identified user corresponding to the input UTOKEN has been removed. Subsequent requests for access to VLF objects with this UTOKEN will fail. **Action**: None.
02	10	**Meaning**: Program error. An unknown UTOKEN was specified. **Action**: Ensure that the user token specified was one received when the user was identified through the COFIDENT macro. Make corrections and rerun the program.
18	00	**Meaning**: Program error. The ALET of the input parameter is not valid. **Action**: Make necessary corrections to ensure that the parameter list ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
28	00	**Meaning**: Environmental error. VLF is not active. **Action**: Issue the START VLF command and rerun the program.
2C	nnnn	**Meaning**: System error. There was an unexpected error in VLF. nnnn is the reason code. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

COFREMOV - List form

Syntax

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

Syntax	Description

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

Parameters

The parameters of the list form are explained as follows:

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

list addr is the name of a storage area to contain the parameters. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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.

COFREMOV - Execute form

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFREMOV
COFREMOV	
ƀ	One or more blanks must follow COFREMOV
,UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).

Syntax	Description
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*retcod*: Rx-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: Rx-type address or register (2) - (12).

Parameters

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

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

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

Chapter 29. COFRETRI — Retrieve a VLF object

Description

The COFRETRI macro enables an application using VLF to obtain a copy of a VLF object on behalf of an end user. Before issuing COFRETRI, or any VLF macro, you need to understand the information on using the virtual lookaside facility (VLF) that appears in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary 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 you issue COFRETRI to retrieve an object on behalf of a user, you must issue COFIDENT to identify the user. COFIDENT relates to COFRETRI in the following ways:

- COFIDENT returns the user token you must supply on COFRETRI.
- COFIDENT establishes the major-name search order for this user.
- COFIDENT defines whether COFRETRI must be issued under a task with a home ASID that matches the home ASID of the issuer of COFIDENT (COFIDENT was issued with SCOPE=HOME), or whether the task invoking COFRETRI can have a different home ASID (COFIDENT was issued with SCOPE=SYSTEM).

Restrictions

None.

Input register information

Before issuing the COFRETRI 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

2-14
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

2-14
Unchanged

15
Used as a work register by the system

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

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFRETRI
COFRETRI	
ƀ	One or more blanks must follow COFRETRI
MINOR=*minor*	*minor*: Rx-type address or register (2) - (12).
,UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).
,TLIST=*tlist*	*tlist*: Rx-type address or register (2) - (12).
,TLSIZE=*tlsize*	*tlsize*: Rx-type address or register (2) - (12).

Syntax	Description
,OBJSIZE=*objsize*	*objsize*: Rx-type address or register (2) - (12).
,CINDEX=*cindex*	*cindex*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).

Parameters

The parameters of the standard form are explained as follows:

MINOR=*minor*
Is a required parameter that identifies the minor name of the object. VLF assumes that the length of the minor name is the same as that specified on the MINLEN parameter when the COFDEFIN macro was issued to define the class. If the class of objects was defined with major name to PDS name correspondence, then the minor name length is 8.

,UTOKEN=*utoken*
Is the required 16-character user token that identifies the user for whom you are retrieving a VLF object. VLF returned the user token when you issued the COFIDENT macro to identify the user to VLF.

,TLIST=*tlist*
Is a required parameter that defines the target area list. The target area list describes target areas into which consecutive areas of the object are to be stored. The target area list consists of a fullword containing the number of target areas, followed by three words for each area:

1. A fullword that contains the ALET that currently addresses the target area. An ALET of 1, referencing the SASN of the caller, or ALETs referencing entries on the PASN access list of the caller, are not allowed.
2. A fullword that contains the 31-bit address of the data for the target area.
3. A fullword that contains the length of the target area.

An address of 0 signifies that VLF is to ignore the specified length; that is, VLF is not to retrieve that part of the object. The maximum number of parts is 16.

,TLSIZE=*tlsize*
Is a required parameter, a fullword that contains the size (in bytes) of the target area list.

,OBJSIZE=*objsize*
Is a required parameter, a fullword that VLF is to use to return the size (in bytes) of the object it retrieves.

,CINDEX=*cindex*
Is a required parameter, a one-byte field that VLF is to use to return the concatenation index of the major name associated with the object it retrieves. The index is the zero-origin relative number of the major name for the object in the major name list of the user retrieving the object. This list is the one that was supplied when the COFIDENT macro identified the user to VLF.

For concatenated partitioned data sets, the CINDEX value is the same as the “K” (concatenation index) value returned when a BLDL macro is issued to locate a member.

,RETCODE=*retcod*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsncod*
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 the COFRETRI macro returns control to your program, GPR 15 (and *retcod*, if you coded RETCODE) contains one of the following hexadecimal return codes. When the COFRETRI macro returns control to your program, GPR 0 (and *rsncod*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Table 35. Return and Reason Codes for the COFRETRI Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: The VLF object was successfully retrieved. OBJSIZE contains the size of the VLF object. CINDEX contains the zero-origin concatenation index number for the object (the zero-origin relative entry number in the major name list supplied on the COFIDENT macro). **Action**: None.
02	00	**Meaning**: Program error. A VLF object has been retrieved that might be the correct object for the user, but the object might also exist in earlier major names in the user's major name list. OBJSIZE contains the size of the VLF object. CINDEX contains the zero-origin concatenation index number for the object (the zero-origin relative entry number in the major name list supplied on the COFIDENT macro). **Action**: Issue the BLDL macro to determine whether the object returned by VLF is the correct object based on the user's major name search order. If the object does exist on DASD in an earlier name in the user's major name search order, then take two steps: • Use the alternate method to acquire the object for the user • Issue the COFCREAT macro to create the VLF object.
04	00	**Meaning**: Program error. The VLF object was retrieved, but the target areas did not receive the entire object. OBJSIZE contains the size of the VLF object. CINDEX contains the zero-origin concatenation index number for the object (the zero-origin relative entry number in the major name list supplied on the COFIDENT service). **Action**: Increase the size of the target area, then issue COFRETRI again.
06	00	**Meaning**: Program error. A VLF object has been retrieved that might be the correct object for the user, but the object might also exist in earlier major names in the user's major name list. Additionally, the target areas did not receive the entire object. OBJSIZE contains the size of the VLF object. CINDEX contains the zero-origin concatenation index number for the object (the zero-origin relative entry number in the major name list supplied on the COFIDENT service). **Action**: Use the same steps as for return code 02 to determine if the object is the correct one. If it is, increase the size of the target area, then issue COFRETRI again.
08	00	**Meaning**: Program error. VLF could not find a matching object to retrieve. **Action**: Use an alternate method to acquire the object for the user. Then issue COFCREAT to create the VLF object.
08	04	**Meaning**: Program error. A retrieve was attempted for a major name that has changed or been deleted. **Action**: Use an alternate method to acquire the object for the user. Then issue COFCREAT to create the VLF object.

Table 35. Return and Reason Codes for the COFRETRI Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
0A	00	**Meaning**: Program error. The parameter list cannot be accessed. **Action**: Make necessary corrections to ensure that the parameter list ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
0C	00	**Meaning**: Program error. The class to which the user is identified is not currently defined. **Action**: Define the class with COFDEFIN and retry the operation.
0E	00	**Meaning**: Program error. The user has insufficient authorization. To retrieve an object for the class, the caller must be a task running in supervisor state or with PKM allowing key 0-7. **Action**: Use an alternate method to acquire the object for the user.
10	00	**Meaning**: Program error. An unknown user token was specified. The most likely reason for this is that the user has been removed from VLF identification because the user's major name list has changed. It is also possible you have not supplied the correct token. **Action**: In either case, you must issue the COFIDENT macro; you must reidentify the user to VLF before you can retrieve objects for the user. Also, ensure that the UTOKEN passed to the COFRETRI macro is valid.
14	00	**Meaning**: Environmental error. VLF incurred a program check when it tried to access the TLIST. You might, for example, have specified a larger target area to VLF than was actually available or specified a target area the user had no authority to modify. **Action**: Rerun the program. If the problem persists, specify a smaller TLSIZE parameter for the TLIST.
18	00	**Meaning**: Program error. **Action**: Ensure that all parameters passed to the COFRETRI macro contain valid data. Make necessary corrections in the application, and rerun the program.
18	02	**Meaning**: Program error. TLSIZE is greater than the maximum allowable size, or the number of target areas is greater than 16. **Action**: Ensure that the first word of the TLIST, which contains the number of target areas, is not greater than 16. Make corrections and rerun the program.
18	0B	**Meaning**: Program error. The object specified on MINOR cannot be accessed using the specified ALET. The ALET is a SASN ALET, or the ALET is not on the dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the MINOR ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
18	0C	**Meaning**: Program error. TLIST cannot be accessed using the specified ALET. The ALET is a SASN ALET, or the ALET is not on the dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the TLIST ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
18	0D	**Meaning**: Program error. A target area in the target list cannot be accessed using the specified ALET. The ALET is a SASN ALET, or the ALET is not on the dispatchable unit access list (DU-AL). **Action**: Make necessary corrections to ensure that the target area ALET is on the dispatchable unit access list (DU-AL) and rerun the program.
28	00	**Meaning**: Environmental error. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

Table 35. Return and Reason Codes for the COFRETRI Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
2C	nnnn	**Meaning**: System error. nnnn is the reason code. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

COFRETRI - List form

Syntax

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

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

Parameters

The parameters of the list form are explained as follows:

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

list addr is the name of a storage area to contain the parameters. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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.

COFRETRI - Execute form

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin name in column 1.
ƀ	One or more blanks must precede COFRETRI
COFRETRI	
ƀ	One or more blanks must follow COFRETRI
MINOR=*minor*	*minor*: Rx-type address or register (2) - (12).
,UTOKEN=*utoken*	*utoken*: Rx-type address or register (2) - (12).
,TLIST=*tlist*	*tlist*: Rx-type address or register (2) - (12).
,TLSIZE=*tlsize*	*tlsize*: Rx-type address or register (2) - (12).
,OBJSIZE=*objsize*	*objsize*: Rx-type address or register (2) - (12).
,CINDEX=*cindex*	*cindex*: Rx-type address or register (2) - (12).
,RETCODE=*retcod*	*retcod*: Rx-type address or register (2) - (12).
,RSNCODE=*rsncod*	*rsncod*: Rx-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: Rx-type address or register (2) - (12).

Parameters

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

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

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

Chapter 30. COFSDONO — Delete a DLF (data lookaside facility) object

Description

Use the COFSDONO macro to cause DLF to delete a DLF object that is no longer needed.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or with PKM allowing key 0-7
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN
AMODE:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space

Programming requirements

Use of Hiperbatch requires real storage.

Restrictions

None.

Input register information

Upon invocation, general purpose registers (GPRs) must contain:

Register
Contents

1
Address of parameter list

13
Address of caller's 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

2-14
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

2-14
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

DLF objects that are no longer needed occupy system resources and should be deleted.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede COFSDONO.
COFSDONO	
ƀ	One or more blanks must follow COFSDONO.
OBJNAME=*name addr*	*name addr*: RX-type address or register (2) - (12).
,RETCODE=*ret addr*	*ret addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn addr*	*rsn addr*: RX-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

OBJNAME=*name addr*
The 64-character name of the DLF object. The name is a 6-character volume serial number followed by 1 to 44-character data set name, left-justified. Pad the 64-character field on the right with blanks (X'40').

,RETCODE=*ret addr*
Specifies the location where the system is to store the return code. The return code is also in general purpose register (GPR) 15. If you specify a storage location, it must be on a fullword boundary.

,RSNCODE=*rsn addr*
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.

,MF=S
Specifies the standard form of the macro. The standard form generates code to put the parameters into an in-line parameter list and invoke the desired service.

ABEND codes

None.

Return and reason codes

When the COFSDONO macro returns control to your program, GPR 15 (and *ret addr*, if you coded RETCODE) contains one of the following hexadecimal return codes. GPR 0 (and *rsn addr*, if you coded RSNCODE) contains one of the following hexadecimal reason codes.

Table 36. Return and Reason Codes for the COFSDONO Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: Successful completion. The DLF object has been deleted. **Action**: None.
02	00	**Meaning**: The object did not exist in DLF. **Action**: Check to see whether the object name is correct.
02	02	**Meaning**: The specified object does not exist in DLF. It has been logically deleted by another routine, or is in the process of being connected or deleted. **Action**: None required.
28	00	**Meaning**: Environmental error. DLF is not active. **Action**: Issue the START DLF command and rerun the job.
2C	nnnn	**Meaning**: System error. There was an unexpected error in DLF. nnnn is the reason code. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

COFSDONO - List form

Syntax

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

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

Parameters

The parameters of the list form are explained as follows:

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

list addr is the name of a storage area to contain the parameters. (If you specify *name* on the macro, the system also equates the name you specify to the same location counter value.)

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.

COFSDONO - Execute form

Syntax

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

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

Syntax	Description
OBJNAME=*name addr*	*name addr*: RX-type address or register (2) - (12).
,RETCODE=*ret addr*	*ret addr*: RX-type address or register (2) - (12).
,RSNCODE=*rsn addr*	*rsn addr*: RX-type address or register (2) - (12).
,MF=(E,*ctrl addr*)	*ctrl addr*: RX-type address or register (2) - (12).

Parameters

Parameters for the execute form of COFSDONO are described in the standard form of the macro with the following exceptions:

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

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

Chapter 31. CONFCHG — Request notification of I/O configuration changes

Description

Use the CONFCHG macro to request notification about dynamic changes in the I/O configuration in your installation. When you invoke CONFCHG with the NOTIFY parameter, you specify whether you want to be notified about one of the following:

- A requested or rejected configuration change that involves deleting a device or deleting a path to a device (CHGREQ parameter)

  Note: IBM recommends that you use the ENFREQ macro (event code 31) instead of CONFCHG.
- A successful configuration change (CHGCOMPL parameter).

  Note: IBM recommends that you use the ENFREQ macro (event code 32) instead of CONFCHG.

When you invoke CONFCHG with the CANCEL parameter, you specify that you no longer want to be notified of changes. You must cancel your NOTIFY request when you no longer want to receive notification.

When you invoke CONFCHG with NOTIFY, you must specify a user-written configuration change exit routine (EXIT parameter). To determine when the exit routine will receive control, you code either the CHGREQ or CHGCOMPL parameter. When an ACTIVATE command is issued, the system ensures that the devices to be deleted are off-line and unallocated. If the activate request has passed this validation step and an authorized program issues CONFCHG CHGREQ, the system passes control to the exit routine. When a requested activation change is rejected, the system also passes control to the exit routine.

If the program issues CONFCHG CHGCOMPL, the system passes control to the exit routine when a dynamic I/O configuration change completes successfully.

When the configuration change exit routine receives control, general purpose register (GPR) 1 contains the address of a parameter list. The parameter list contains information about the change that occurred, such as the specific device that is being added, modified, or deleted. See *z/OS MVS Programming: Authorized Assembler Services Guide* for complete information on coding the configuration change exit routine.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state, with 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

Restrictions

None.

Register information

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

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

Programming requirements

The caller of CONFCHG must ensure that the configuration change exit routine resides in common storage. Before coding CONFCHG with the EXIT parameter, the caller must set to 1 the high-order bit of the exit routine's address.

Performance implications

None.

Syntax

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

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

Syntax	Description
CONFCHG	
ƀ	One or more blanks must follow CONFCHG
NOTIFY	
CANCEL	
,CHGREQ	
,CHGCOMPL	
,TOKEN=*token addr*	*token addr*: RS-type address or register (2) - (12). Required with NOTIFY only if you plan to cancel the request upon completion. Required with CANCEL.
,EXIT=*exitrtn addr*	*exitrtn addr*: RS-type address or register (2) - (12). Required only with NOTIFY. Not valid with CANCEL.
,RETCODE=*rc addr*	*rc addr*: RX-type address or register (2) - (12).
,MF=S	

Parameters

The parameters are explained as follows:

NOTIFY
CANCEL
The required parameter that requests:

- Notification of I/O configuration changes (NOTIFY), or
- Cancellation of a previous notification request (CANCEL).

,CHGREQ
,CHGCOMPL
The required parameter that specifies whether the caller wants notification of:

- Requested or rejected I/O configuration changes that involve deleting a device or deleting a path to a device (CHGREQ)
- I/O configuration changes that completed successfully (CHGCOMPL).

,TOKEN=*token addr*
For NOTIFY, specifies a 4-character output field into which the system returns a token to identify the request. TOKEN is required with NOTIFY only if you plan to cancel your notification request when completed.

For CANCEL, specifies the 4-character token returned by NOTIFY to cancel a specific notification request. TOKEN is required with CANCEL.

,EXIT=*exitrtn addr*
The required parameter (for NOTIFY) that specifies the address of the configuration change exit routine to receive control. The exit routine must reside in common storage, and the high-order bit of the routine's address must be set to 1. Do not code this parameter with CANCEL.

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

,MF=S
An optional parameter that specifies the standard form of the macro. The standard form places the parameters into an in-line parameter list and invokes the service. The system checks for required parameters and supplies optional parameters that are not specified. MF=S is the default.

Return codes

When control returns from CONFCHG, GPR 15 (and *rc addr*, if you coded RETCODE) contains one of the following return codes:

Table 37. Return Codes for the CONFCHG Macro

Hexadecimal Return Code	Meaning
00	**Meaning**: CONFCHG processing completed successfully.
04	**Meaning**: Duplicate CONFCHG NOTIFY request.
08	**Meaning**: Error in the control parameter list.
10	**Meaning**: Error in CONFCHG processing.
14	**Meaning**: System is not able to process request.
18	**Meaning**: System cannot obtain storage for the request.
1C	**Meaning**: Token for CANCEL request is not valid.

Example 1

Issue the CONFCHG macro so that the user exit, CHGEXIT, gets control when a configuration change completes.

```
CONFCHG NOTIFY,CHGCOMPL,EXIT=EXIT_ADD,TOKEN=TOKEN
```

Example 2

Use the CONFCHG macro to indicate the user exit, CHGEXIT, should not be called after configuration changes.

```
CONFCHG CANCEL,CHGCOMPL,TOKEN=TOKEN
```

CONFCHG - List form

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.

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

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

Parameters

The parameter is explained as follows:

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

CONFCHG - Execute form

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

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

Syntax	Description
CONFCHG	
ƀ	One or more blanks must follow CONFCHG.
NOTIFY	
CANCEL	
,CHGREQ	
,CHGCOMPL	
,TOKEN=*token addr*	*token addr*: RX-type address or register (2) - (12). Required with NOTIFY only if you plan to cancel the request upon completion. Required with CANCEL.
,EXIT=*exitrtn addr*	*exitrtn addr*: RX-type address or register (2) - (12). Required only with NOTIFY. Not valid with CANCEL.
,RETCODE=*rc addr*	*rc addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,MF=(E,*list addr*,COMPLETE)	**Default:** COMPLETE

Parameters

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

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

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

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

Chapter 32. CPF — Manage a command prefix

Description

The CPF (command prefix facility) macro allows you to manage command prefixes. A command prefix enables an operator to enter a command from a system in a sysplex, and route that command to the appropriate subsystem for execution. The CPF macro allows an application to use command prefixes to associate an operator command with a "target" system. The command prefixes are available to any system in the sysplex.

Use the CPF macro to:

- Define a new command prefix
- Delete an existing command prefix
- Redefine an existing command prefix for a system or owner name.

The macro has a list and an execute form, but no standard form. The parameters are explained in detail on the execute form of the macro.

For more information on the CPF macro, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

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

Programming requirements

None.

Restrictions

Prefixes cannot be supersets or subsets of existing prefixes. See *z/OS MVS Programming: Authorized Assembler Services Guide* for details about defining valid prefixes.

Input register information

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

Register
Contents

13
The address of an 18-word save area

Output register information

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

Register
Contents

0
Reason code, unless you receive return code X'0C'. In this case, register 0 contains a system completion code or zero.

1
Used as a work register by the system.

2-13
Unchanged

14
Used as a work register by the system.

15
Return code.

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

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.

CPF - List form

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CPF.
CPF	
ƀ	One or more blanks must follow CPF.
MF=(L,*list addr*)	*list addr*: RX-type address or register 2-12.

The parameters are explained as follows:

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

list addr is the name of the storage area to contain the parameters.

CPF - Execute form

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CPF.
CPF	
ƀ	One or more blanks must follow CPF.
	Valid parameters (Required parameters are underlined):
REQUEST=DEFINE	PREFIX, OWNER, SCOPE, FAILDISP, REMOVE
REQUEST=DELETE	PREFIX, CURSYS
REQUEST=REDEFINE	PREFIX, OWNER, CURSYS, NEWSYS

Syntax	Description
,PREFIX=*prefix addr*	*prefix addr*: RX-type address or register 2-12.
,OWNER=*owner addr*	*owner addr*: RX-type address or register 2-12.
,SCOPE=SYSPLEX	**Default:** SCOPE=SYSPLEX
,SCOPE=SYSTEM	
,FAILDISP=PURGE	**Default:** FAILDISP=PURGE
,FAILDISP=SYSPURGE	
,FAILDISP=RETAIN	
,REMOVE=NO	**Default:** REMOVE=NO
,REMOVE=YES	
,CURSYS=*sys name*	*sys name*: RX-type address or register 2-12.
,NEWSYS=*sys addr*	*sys addr*: RX-type address or register 2-12.
,MF=(E, *list addr*)	*list addr*: RX-type address or register 2-12

The parameters are explained as follows:

REQUEST=DEFINE
REQUEST=DELETE
REQUEST=REDEFINE
Specifies the desired command prefix facility function to be performed. You can specify only one function at a time. The three functions are:

DEFINE
Creates the definition for a new command prefix.

DELETE
Deletes an existing command prefix.

REDEFINE
Defines a new receiving system for a given prefix, and a new owner name, if needed.

,PREFIX=*prefix addr*
Specifies the address of a required 8-byte field containing the command prefix. If the prefix is less than 8 characters, it must be left-justified and padded with blanks.

,OWNER=*owner addr*
Specifies the address of an 8-byte field containing a name that identifies the subsystem owning the command prefix (for example, JES2, JES3, IMS). If the name is less than 8 characters, it must be left-justified and padded with blanks.

,SCOPE=SYSPLEX
,SCOPE=SYSTEM
Specifies the range of systems to which a command with this prefix can be routed for execution. The values are:

SYSPLEX
The command issued can be routed to another system in the sysplex for execution. If SCOPE is not specified, this is the default.

Note: If the installation has defined the security profile MVS.CPF.ROUTE.CHECK in the OPERCMDS class, the issuer of the command requires sufficient authority to the MVS.ROUTE.CMD.system to route the command to a different system in the sysplex.

SYSTEM
The command issued will execute in the system on which the command is entered.

,FAILDISP=PURGE
,FAILDISP=SYSPURGE
,FAILDISP=RETAIN
Specifies the failure disposition of the prefix being defined. Any one of the following can be specified:

PURGE
The command prefix is automatically deleted when the receiving system is removed from the sysplex, or the defining address space terminates. If the FAILDISP is not specified, this value is the default.

SYSPURGE
The command prefix is automatically deleted when the receiving system is removed from the sysplex, but not when the defining address space terminates.

RETAIN
The command prefix will persist even if a system is removed or the address space of the routine processing the command terminated. In this case, the owning subsystem is responsible for redefining the command prefix for another system or deleting the command prefix, respectively.

,REMOVE=YES
,REMOVE=NO
Specifies whether the command prefix is removed from the command text prior to being executed on the receiving system. REMOVE=NO indicates the command prefix and the command are presented to the receiving system. If the REMOVE parameter is not specified, this is the default. REMOVE=YES indicates the command prefix is removed from the command before it is presented to the receiving system.

,CURSYS=*sys name*
Specifies the address of an 8-byte field containing the name of the system for which the prefix was defined. The system is the system on which the command will be processed. Issue the DISPLAY XCF command to obtain a list of the names of systems in the sysplex. If the system name is less than 8 bytes, it must be left-justified and padded on the right with blanks. The default is the name of the system on which the CPF macro is invoked.

,NEWSYS=*sys addr*
Specifies the address of an 8-byte field containing the name of the new system to which commands with this prefix should be routed in the event that the system specified on CURSYS fails. If the system name is less than 8 bytes, it must be left-justified and padded on the right with blanks. The default is the name of the system on which the CPF macro is invoked.

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

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

ABEND codes

None.

Return and reason codes

When the CPF macro with REQUEST=DEFINE returns control to your program, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code.

Table 38. Return and Reason Codes for the CPF Macro with REQUEST=DEFINE

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: CPF completed successfully. **Action**: None.
00	04	**Meaning**: Environmental error. You specified DEFINE with SCOPE=SYSPLEX, but the system was in XCF-local mode. **Action**: Ensure that you are running in a sysplex, or change the SCOPE parameter. Retry the request.
04	04	**Meaning**: Program error. The prefix contains characters that are not supported. For a list of supported characters, see Assigning a prefix in *z/OS MVS Programming: Authorized Assembler Services Guide*. **Action**: Correct the prefix and retry the request.
04	08	**Meaning**: Program error. The OWNER parameter on the DEFINE request contained characters not in the range of X'41' to X'FE'. **Action**: Correct the OWNER parameter and retry the request.
08	08	**Meaning**: Program error. You specified DEFINE for a prefix that already exists. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: If you specified the wrong prefix, correct the problem and retry the request.
08	0C	**Meaning**: Program error. You specified DEFINE with a prefix that is a subset of an existing prefix. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Refer to prefix subset requirements. Correct the problem and retry the request.
08	10	**Meaning**: Program error. You specified DEFINE with a prefix that was a superset of an existing prefix. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Refer to prefix subset requirements. Correct the problem and retry the request.
0C	None.	**Meaning**: System error. A broadcast of an updated CPF table failed, or an abend occurred. **Action**: If an abend occurred, register 0 contains the abend code. Record the return code and supply it to the appropriate IBM support personnel.

When the CPF macro with REQUEST=DELETE returns control to your program, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code.

Table 39. Return and Reason Codes for the CPF Macro with REQUEST=DELETE

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	00	**Meaning**: CPF completed successfully. **Action**: None.
04	04	**Meaning**: Program error. The prefix contains characters that are not supported. For a list of supported characters, see Assigning a prefix in *z/OS MVS Programming: Authorized Assembler Services Guide*. **Action**: Correct the prefix and retry the request.

Table 39. Return and Reason Codes for the CPF Macro with REQUEST=DELETE (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
08	04	**Meaning**: Environmental error. You specified DELETE, but the prefix was not found in the CPF table. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Correct the problem and retry the request.
08	1C	**Meaning**: Program error. You specified DELETE, but no CPF table exists. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Determine whether the CPF table should exist.
0C	None.	**Meaning**: System error. A broadcast of an updated CPF table failed, or an abend occurred. **Action**: If an abend occurred, register 0 contains the abend code. Record the return code and supply it to the appropriate IBM support personnel.

When the CPF macro with REQUEST=REDEFINE returns control to your program, GPR 15 contains a hexadecimal return code and GPR 0 contains a hexadecimal reason code.

Table 40. Return and Reason Codes for the CPF Macro with REQUEST=REDEFINE		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
00	00	**Meaning**: CPF completed successfully. **Action**: None.
04	04	**Meaning**: Program error. The prefix contains characters that are not supported. For a list of supported characters, see Assigning a prefix in *z/OS MVS Programming: Authorized Assembler Services Guide*. **Action**: Correct the prefix and retry the request.
04	08	**Meaning**: Program error. The OWNER parameter on the DEFINE request contained characters not in the range of X'41' to X'FE'. **Action**: Correct the OWNER parameter and retry the request.
04	0C	**Meaning**: Program error. You specified REDEFINE for a prefix that was defined with FAILDISP=PURGE. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Correct the problem and retry the request.
08	04	**Meaning**: Environmental error. You specified REDEFINE, but the prefix was not found in the CPF table. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Correct the problem and retry the request.
08	14	**Meaning**: Program or environmental error. You specified REDEFINE, but the NEWSYS parameter specified a system not in the sysplex. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Wait for the specified system to join the sysplex, or determine if you specified an incorrect system name. Make any necessary corrections and retry the request.
08	18	**Meaning**: Program error. You redefined a prefix and targeted it for another system in the sysplex. However, that system had the same prefix already defined. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Correct the problem and retry the request.

Table 40. Return and Reason Codes for the CPF Macro with REQUEST=REDEFINE (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
08	1C	**Meaning**: Program error. You specified REDEFINE, but no CPF table exists. CPF internally issues the DISPLAY OPDATA command which displays the command prefixes defined for subsystems in the sysplex. **Action**: Determine whether the CPF table should exist.
0C	None.	**Meaning**: System error. A broadcast of an updated CPF table failed, or an abend occurred. **Action**: If an abend occurred, register 0 contains the abend code. Record the return code and supply it to the appropriate IBM support personnel.

Example

Define a prefix that causes all commands issued with that prefix to be sent to system *cvtsname* for processing.

```
CPF MF=(L,CPFLIST)
    ⋮
CPF REQUEST=DEFINE,
    PREFIX=CVTSNAME,
    OWNER=OWNER,
    SCOPE=SYSPLEX,
    FAILDISP=PURGE,
    REMOVE=YES,
    MF=(E,CPFLIST)
    ⋮
    OWNER DC CL8'CONSOLE '
    ⋮
```

Chapter 33. CPOOL — Perform cell pool services

Description

The CPOOL macro performs the following functions:

- Creates a cell pool (BUILD)
- Obtains a cell from the pool (GET, COND)
- Returns a cell to the cell pool (FREE)
- Requests contraction of a contractible cell pool (CONTRACT)
- Deletes a previously built cell pool (DELETE)
- Places the starting and ending addresses of the cell pool extents in a buffer (LIST).

The CPOOL macro is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP* with the exception of the TCB, LINKAGE, OWNER, and VERIFY parameters.

Before obtaining storage, be sure to read the information on subpools in "Virtual Storage Management" in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

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, **one or more** of the following: – Supervisor state – PSW key 0-7 – APF authorization – A PSW key mask (PKM) that allows the calling program to switch its PSW key to match the key of the storage to be obtained or released • For other subpools, CPOOL CONTRACT, the TCB parameter, and the MULTIHDR=YES parameter, **one or more** of the following: – Supervisor state – PSW key 0-7, PSW key 0 for TCB parameter – APF authorization • For LINKAGE=BRANCH, supervisor state and key 0 • For the VERIFY parameter, supervisor state
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN.
AMODE:	For GET with MULTIHDR=YES and FREE with MULTIHDR=YES, 31-bit. Otherwise, 24- or 31-bit.
ASC mode:	Primary. For LIST requests and LINKAGE=BRANCH, primary or secondary.

Environmental factor	Requirement
Interrupt status:	• For private (local) and pageable common (global) storage requests, the caller must be enabled for I/O and external interrupts. • For GET,UNCOND requests, the caller must not be disabled when the specified cell pool is in a disabled reference (DREF) subpool. • For all other requests, enabled or disabled for I/O and external interrupts.
Locks:	The following locks must be held by the caller or must be obtainable by CPOOL: • For private storage or for pageable common: – 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. – CMS lock. • For other storage (DREF or fixed common), the caller may hold locks, but is not required to hold any.
Control parameters:	Must reside in the caller's primary address space. Except for TCB, parameters 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 by 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, unchanged. Otherwise, used as work registers by the system.

5-13
If LINKAGE=SYSTEM, REGS=SAVE, COND REGS=USE, or MULTIHDR=YES is specified, unchanged. Otherwise, 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, 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 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,MULTIHDR=NO, unchanged. Otherwise, used as a work register by the system.

1
For CPOOL GET,COND,MULTIHDR=NO, 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,MULTIHDR=NO or CPOOL FREE,MULTIHDR=NO, unchanged. Otherwise, used as a work register by the system.

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

Performance implications

The CPOOL macro offers better performance than GETMAIN–FREEMAIN and STORAGE for obtaining and releasing many identically sized storage areas.

Syntax

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):

Syntax	Description
BUILD	CSIZE, PCELLCT, SCELLCT, SP, BNDRY, LOC, CPID, KEY, TCB, HDR, LINKAGE, OWNER, MULTIHDR, MAXCELLS, CELLSPERCPU, CELLSHARE, CONTRACTIBLE, AUTOCONTRACT, FREECELLS, FREECELLSDIVISOR, CONTINTVL, FREECELLSPERCPU
GET	CPID, UNCOND, COND, CELL, REGS, LINKAGE, MULTIHDR, CONTRACTIBLE
FREE	CPID, CELL, REGS, MULTIHDR, CONTRACTIBLE
CONTRACT	CPID, FREECELLS
DELETE	CPID, LINKAGE
LIST	CPID, WORKAREA, VERIFY
,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
,BNDRY=QWORD	The default value depends on the specified CSIZE value. If CSIZE is a multiple of 8, cells reside on doubleword 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.
,LOC=24	**Default**: LOC=RES
,LOC=31	
,LOC=(31,31)	
,LOC=(31,64)	
,LOC=(31,PAGEFRAMESIZE1MB)	
,LOC=RES	
,LOC=(RES,31)	

Syntax	Description
,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 (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 list of subpool characteristics in *z/OS MVS Programming: Authorized Assembler Services Guide* for information on storage keys for specific subpools.
,TCB=*tcb addr*	*tcb addr*: RX-type address or register (0), (2) - (12). **Default:** TCB address in PSATOLD.
,HDR=*hdr*	*hdr*: Character string enclosed in single quotation marks, RX-type address, or register (0), (2) - (12). **Default:** `'CPOOL CELL POOL'`
,LINKAGE=SYSTEM	**Default:** LINKAGE=SYSTEM
,LINKAGE=BRANCH	**Note:** Do not specify LINKAGE with FREE or LIST requests or the GET request with the COND parameter.
,REGS=SAVE	**Default:** REGS=SAVE
,REGS=USE	
,WORKAREA=*(workarea,length)*	*workarea*: Symbol, RX-type address, or register (0), (2) - (12). *length*: Symbol or decimal number.
,VERIFY=NO	**Default:** VERIFY=NO
,VERIFY=YES	
,OWNER=HOME	**Default:** OWNER=HOME
,OWNER=PRIMARY	
,OWNER=SYSTEM	

Syntax	Description
,MULTIHDR=NO	**Default:** MULTIHDR=NO
,MULTIHDR=YES	
,MULTIHDR=COND	
,MAXCELLS=*mmmm*	
,CELLSPERCPU=*nnnn*	
,CELLSHARE=NO	**Default:** CELLSHARE=NO
,CELLSHARE=YES	
,CONTRACTIBLE=NO	**Default:** CONTRACTIBLE=NO
,CONTRACTIBLE=COND	
,AUTOCONTRACT=YES	**Default:** AUTOCONTRACT=YES
,AUTOCONTRACT=NO	
,FREECELLS=*free-cells*	*free-cells*: Decimal number, RS-type address, or register (2) - (12).
,FREECELLS=CPOOL_BUILD	
,FREECELLDIVISOR=*divisor*	*divisor*: Decimal number, RS-type address, or register (2) - (12).
,CONTINTVL=*seconds*	*seconds*: Decimal number, RS-type address, or register (2) - (12).
,FREECELLSPERCPU=(*xxxx*,...,*zzzz*)	*xxxx*,...,*zzzz*: One or more of: NO, ALL, STANDARD, , ZAAP, ZIIP.
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=0	
,PLISTVER=1	

Parameters

The parameters are explained as follows:

BUILD
GET
FREE
CONTRACT
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.

GET
Attempts to obtain a cell from the previously built cell pool. This request can be conditional or unconditional as described under the UNCOND and COND keywords.

FREE
Returns a cell to a cell pool.

CONTRACT
Requests contraction of a previously built cell pool. This request is valid only for cell pools built with MULTIHDR=YES and CONTRACTIBLE=COND, and requires the CPID and FREECELLS parameters.

The FREECELLS parameter identifies a fullword that indicates that the contraction is to end with no more than *n* free cells. FREECELLS=CPOOL_BUILD indicates that you want to use the FREECELLS, FREECELLSDIVISOR, and FREECELLSPERCPU specifications that were provided on the CPOOL BUILD request to perform the contraction.

When providing a value for FREECELLS, you can use a specific number of free cells, or you can calculate the number based on the number of online CPUs in certain fields in the CSD area (mapped by IHACSD), such as:

CSD_NUMBER_ONLINE_CPUS
CSD_NUMBER_ONLINE_STANDARD_CPS
CSD_NUMBER_ONLINE_IFAS
CSD_NUMBER_ONLINE_ZIIPS

If transactional execution is not available, this request is effectively a no-op (and may return with a warning indicating that contraction is not available).

Your program should issue a CONTRACT request periodically (perhaps every few minutes). If there are fewer free cells than the request indicates, no contraction is performed. Contraction processing preferentially frees cells from processors that have used no cells for a certain duration. If this does not result in freeing enough cells, processing continues to free one cell per CPU that has a free cell until enough cells have been freed. This process continues in a loop, perhaps revisiting CPUs that have been processed on a previous iteration, until enough cells have been freed.

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.

For a contractible cell pool, any extent about which CPOOL LIST returns information might no longer exist by the time the caller examines the data if a contract operation had occurred (either by CPOOL CONTRACT or as a result of automatic contract processing within CPOOL itself). The caller can prevent this by ensuring that no GET, FREE, or CONTRACT operations occur.

The caller can prevent CPOOL contract processing from running by holding the proper system lock, as described in Table 41 on page 264; contract processing will not proceed for the cell pool until the lock is released.

Table 41. Locks required to prevent CPOOL contract processing

If the cell pool resides in...	The caller must...
Private storage	Hold the local lock of the current primary address space.
Pageable common area	Hold the CMS lock (which, in turn, requires holding a local lock).

Non-pageable common area	1. Issue the following request before calling CPOOL LIST:
	`SETLOCK OBTAIN,TYPE=VSMFIX,MODE=UNCOND`
	2. Issue the following request after calling CPOOL LIST:
	`SETLOCK RELEASE,TYPE=VSMFIX`

If your program is covered by a functional recovery routine (FRR) and the FRR receives control after SETLOCK OBTAIN has been issued, the FRR must either issue SETLOCK RELEASE,TYPE=VSMFIX or must issue the SETLOCK and SETRP macros, as follows:

```
   SETLOCK TEST,TYPE=VSMFIX,BRANCH=(NOTHELD,NOVSMFIX)
   SETRP FRELOCK=VSMFIX
NOVSMFIX DS 0H
```

Notes:

1. You can use the REGS parameter on the SETLOCK macro if you do not want the default behavior.
2. You can use the WKAREA parameter on the SETRP macro if you do not want the default behavior.

,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 CPOOL service routine returns to the caller without a cell. The CPOOL service routine places a zero in the field that contains the address of the newly obtained cell.

If you specify UNCOND or U and no more free cells are available in the cell pool, the CPOOL service routine obtains more storage for the cell pool. CPOOL then obtains a new cell for the caller. An unconditional CPOOL GET request fails only if enough storage is not available to extend the cell pool.

,PCELLCT=*primary cell count*

Specifies the number of cells expected to be needed in the initial extent of the cell pool.

,SCELLCT=*secondary cell count*

Specifies the number of cells expected to be in each secondary or noninitial extent of the cell pool.

,CSIZE=*cell size*

Specifies the number of bytes in each cell of the cell pool. If CSIZE is a multiple of 8, the cell resides on doubleword boundaries. If CSIZE is a multiple of 4, the cell resides on word boundaries. The minimum value of CSIZE is 4 bytes.

When the specified cell size is less than 256 bytes, the number of elements allocated to an extent may be more than what is expected. The extent might also 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. See the list of subpool characteristics in *z/OS MVS Programming: Authorized Assembler Services Guide* for information on authorization requirements pertaining to specific subpools.

,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.

Notes:

1. When BNDRY=DWORD is explicitly specified, a CSIZE value that is multiple of 8 must also be specified to ensure that each cell is on at least a doubleword boundary.
2. When BNDRY=QWORD is explicitly specified, a CSIZE value that is multiple of 16 must also be specified to ensure that each cell is on at least a quadword boundary.

,LOC=24
,LOC=31
,LOC=(31,31)
,LOC=(31,64)
,LOC=(31,PAGEFRAMESIZE1MB)
,LOC=RES
,LOC=(RES,31)
,LOC=(RES,64)

Specifies the location of virtual storage and central storage for the cell pool. The location of central storage using this parameter is guaranteed only after the storage is fixed.

LOC=24 indicates that central and virtual storage are to be located below 16 megabytes. LOC=24 must not be used to allocate disabled reference (DREF) storage.

Note: Specifying LOC=BELOW is the same as specifying LOC=24. LOC=BELOW is still supported, but IBM recommends using LOC=24 instead.

LOC=31 and LOC=(31,31) indicate that virtual and central storage can be located anywhere below 2 gigabytes.

Note: Specifying LOC=ANY or LOC=(ANY,ANY) is the same as specifying LOC =31 or LOC=(31,31). LOC=ANY and LOC=(ANY,ANY) are still supported, but IBM recommends using LOC=31 or LOC=(31,31) instead.

LOC=(31,64) indicates that virtual storage is to be located below 2 gigabytes and central storage can be located anywhere in 64-bit storage.

LOC=(31,PAGEFRAMESIZE1MB) indicates that virtual storage is to be located below 2 gigabytes and central storage can be backed anywhere in 64-bit storage, preferably by 1 megabyte page frames. When specifying LOC=(31,PAGEFRAMESIZE1MB) during CPOOL BUILD:

- The only *xx* sub-parameter value that can be validly specified in combination with the PAGEFRAMESIZE1MB *yy* sub-parameter on the LOC statement of the CPOOL BUILD macro is 31:

```
LOC=(31,PAGEFRAMESIZE1MB)
```

- PAGEFRAMESIZE1MB indicates that the preferred page frame size for the CPOOL virtual storage range is 1 MB.

  Attention: PAGEFRAMESIZE1MB is a page size preference only; it does not guarantee that the virtual storage range will be backed by large pages.

- There are no requirements that the 31-bit virtual storage obtained be large page aligned or that it be a multiple of the specified large page size.
- The LOC(31,PAGEFRAMESIZE1MB) parameter has no effect on other parameters that can be specified on CPOOL BUILD requests.
- Subpools that support backing by 1 megabyte page frames are identified in Table 8-1 in *z/OS MVS Diagnosis: Reference*.

LOC=RES indicates that the location of virtual and central storage depends on the location of the caller. If the caller resides below 16 megabytes, virtual and central storage are to be allocated below 16 megabytes; if the issuer resides above 16 megabytes, virtual and central storage can be located anywhere.

LOC=(RES,31) indicates that the location of virtual storage depends upon the location of the caller. If the caller resides below 16 megabytes, virtual storage is to be located below 16 megabytes; if the caller resides above 16 megabytes, virtual storage can be located anywhere below 2 gigabytes. In either case, central storage can be located anywhere below 2 gigabytes.

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 4-byte address or register that is to contain (BUILD request) or contains (DELETE, FREE, GET, and LIST requests) a cell pool identifier. The system returns this identifier to the caller after the issuer creates the cell pool using CPOOL BUILD. The issuer must specify the same CPID on subsequent DELETE, FREE, GET, and LIST requests.

,CELL=*cell addr*
Specifies the 4-byte address or register that is to contain (GET request) or contains (FREE request) the cell pool address.

,KEY=*key number*
Specifies the storage key in which storage is to be obtained. If a register is specified, the storage key is taken from bits 28-31. This parameter is valid for subpools 129-132, 227-231, 241, and 249.

,TCB=*tcb addr*
Specifies the address of the input TCB, which the system uses to assign ownership of private storage. The TCB must be within the currently addressable address space. If the caller specifies zero as the TCB address, the CPOOL service routine uses the TCB address in ASCBXTCB. If the CPOOL request is for private storage and the caller does not specify TCB, the default is the TCB address in PSATOLD.

For an explanation of the term *input TCB*, and to determine the system-assigned defaults for ownership of private storage, see the topic on selecting the subpool for your virtual storage request in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Note: The TCB resides in storage below 16 megabytes.

,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.

,LINKAGE=SYSTEM
,LINKAGE=BRANCH
Specifies the type of linkage used in CPOOL processing:

LINKAGE=SYSTEM
The linkage uses a non-SVC-entry.

LINKAGE=BRANCH
The linkage uses branch entry.

,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 the 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, to complete a CPOOL LIST request, your program might have to issue CPOOL LIST more than once. If CPOOL LIST

uses up all the space in the work area, but still has more information to return, it indicates (with a return code) that there are more address pairs. Your program can then reissue CPOOL LIST to get more information, and keep reissuing CPOOL LIST until all of the information is returned.

CPOOL LIST must be able to tell the difference between the beginning of a request (that is, the first time your program issues CPOOL LIST to get some information about a cell pool) and the continuation of a request (that is, when your program issues CPOOL LIST to get more information). Your program tells CPOOL LIST that it is beginning a new request by setting the first bit of word 0 in the work area to 1.

Until your program has obtained all the information about a cell pool that it needs from CPOOL LIST, it should not change the setting of that bit, nor should it issue a GET, FREE, or DELETE request for that cell pool. (If your program does issue a GET or FREE request before it has obtained all of the information it needs from CPOOL LIST, it must begin a new CPOOL LIST request; that is, set the first bit of word 0 to 1 and start all over again. If your program deletes the cell pool, it can no longer issue the CPOOL LIST for that cell pool.)

CPOOL LIST uses the second through fourth words (words 1-3) in the work area to return information to your program:

- Word 1 contains the return code. See "Return codes" on page 272
- 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.

,VERIFY=NO
,VERIFY=YES

To make sure the virtual storage control blocks are backed by central storage and accessible, specify VERIFY=YES. The default is VERIFY=NO.

,OWNER=HOME
,OWNER=PRIMARY
,OWNER=SYSTEM

Specifies the entity to which the system will assign ownership of requested CSA, ECSA, SQA, and ESQA storage. The system uses this ownership information to track the use of CSA, ECSA, SQA and ESQA storage. This parameter can have one of the following values:

HOME
The home address space.

PRIMARY
The primary address space.

SYSTEM
The system (the storage is not associated with an address space). Specify this value if you expect the requested storage to remain allocated after termination of the job that obtained the storage.

The default value is OWNER=HOME. The system ignores the OWNER keyword unless you specify a CSA, ECSA, SQA or ESQA subpool on the SP parameter.

Storage tracking is available as of MVS/SP Release 4.3. Programs that issue the CPOOL macro with the OWNER=PRIMARY or OWNER=SYSTEM parameter must run on MVS/SP 4.3 or later. However, programs that issue the CPOOL macro with the OWNER=HOME parameter can run on any system.

Note: For CPOOL GET, the system determines the owning address space at the time of the GET request, even if you specify the address space when you issue CPOOL BUILD. For example, if CPOOL BUILD specifies OWNER=HOME with PASN=HASN=5, and CPOOL GET is issued with HASN=8 and PASN=5, the owner for the GET is address space 8. Therefore, if your cross-memory environment is different for CPOOL GET and CPOOL BUILD, you should ensure that the correct owning address space is specified.

,MULTIHDR=NO
,MULTIHDR=YES
,MULTIHDR=COND

Specifies whether a multi-header cell pool is to be created. The default is MULTIHDR=NO.

References to MULTIHDR=YES are intended also to include the case of MULTIHDR=COND when contraction is available.

NO
Specifies not to use a multi-header cell pool.

YES
On a BUILD request, specifies that a cell pool with multiple headers is to be created. Only authorized callers are supported (system key, supervisor state, or APF-authorized). A header is created for each CPU up to the maximum number of CPUs that are supported on the system (CVTMAXMP + 1). These headers are contiguous in storage. Each header is the same size as a CPU cache line as specified in ECVTCACHELINESIZE.

PCELLCT and SCELLCT are not supported with MULTIHDR=YES. Additionally, MULTIHDR=YES is not supported on a GET request when LINKAGE=BRANCH is specified

When specified on a GET REQUEST, LINKAGE= is not supported. Each MULTIHDR=YES allocated cell has a 16-byte prefix area that is reserved by the system for internal system usage. A GET or FREE MULTIHDR=YES invocation is only supported for 31-bit Amode callers.

Use of GET and FREE for a CPOOL built with MULTIHDR=YES are also limited to authorized callers, even if the GET or FREE request does not specify the MULTIHDR keyword. A non-authorized caller may get a C78 abend on a CPOOL operation for a CPOOL built with MULTIHDR=YES.

If you specify MULTIHDR=YES or MULTIHDR=COND on the BUILD request for a cell pool, you must make the same specification on every CPOOL GET or CPOOL FREE request for that cell pool. To maintain performance efficiency, the system does not enforce this.

COND
On a GET or FREE request, specifies that if contraction is not available (via a runtime check), then use MULTIHDR=NO; otherwise, use MULTIHDR=YES. MULTIHDR=COND is valid only for cell pools that were built with MULTIHDR=YES and CONTRACTIBLE=COND. Your program must have provided similar dual-path logic for the CPOOL BUILD request, checking the PSATXC bit or CVTTXC bit, along with the z/OS V2R2 feature bit (CVTzOS_V2R2), if not known to be running on a system with z/OS V2R2 functions available.

,MAXCELLS=*mmmm*
When specified on a BUILD,MULTIHDR=YES request, this parameter specifies the maximum number of cells that are to be allocated to the cell pool. If this keyword is not specified, the default value of 0 is used, which indicates that no maximum exists for the cellpool. The syntax for MAXCELLS= is identical to that of SCELLCT=. A negative value will result in a C78-20 abend, similar to what occurs for PCELLCT and SCELLCT.

This parameter is applicable only if the caller subsequently does a conditional GET request specifying MULTIHDR=YES. The GET processing expands the cell pool conditionally based on the value of MAXCELLs. An 0 cell address is returned if the allocated cells in the cell pool have reached this maximum value and no cells are available. If the value of MAXCELLS is not specified, GET,COND will function identically to GET,UNCOND and will pool.

The maximum number of cells allocated to a cell pool can be exceeded by up to CELLSPERCPU-1 cells if MAXCELLS is not a multiple of CELLSPERCPU.

,CELLSPERCPU=*nnnn*
When specified on a BUILD,MULTIHDR=YES request, this parameter specifies the number of cells to be allocated in a CPU extent. The syntax for CELLSPERCPU is identical to that of SCELLCT. A negative value will result in a C78-20 abend. A value that results in a too large extent value will result in a C78-A4 abend. This is similar to what occurs for SCELLCT and PCELLCT. The value specified is the number of cells to be allocated in the CPU extent for the CPU requesting the cell when GET expands the cell pool. If this is not specified, the default value of one is used for the cells to be allocated a CPU extent.

If MAXCELLS is specified and is not a multiple of CELLSPERCPU, then the maximum number of allocated cells in a cell pool can be exceeded by up to CELLSPERCPU-1 cells.

,CELLSHARE=NO
,CELLSHARE=YES

When specified on a BUILD,MULTIHDR=YES request, specify this parameter to allow free cells from a cell pool with multiple headers to be shared by CPUs that are requesting for free cells. Note that a free cell might be accessible by only some of the CPUs.

When CELLSHARE=YES is specified, note that:

- Using CELLSHARE to share free cells between CPUs can help balance CPUs. Some CPUs, for example, might accumulate excessive cells because of a spike in usage. Other CPUs can use some of the accumulated excessive cells to expand their pool of available cells without having to issue a GETMAIN request.
- If you specified MAXCELLS for a cell pool with multiple headers and the MAXCELLS limit on the number of cells allocated for a cell pool has been reached, sharing of free cells between neighboring CPUs occurs automatically, regardless of what you specify for CELLSHARE.
- Any cell pool with multiple headers can benefit from cell sharing. However, the cell pools that benefit the most are the ones that are expected to use a great many cells and do not have the maximum cell number limit specified by the MAXCELLS parameter. This is because MAXCELLS caps the number of cells a cell pool can use.

,CONTRACTIBLE=NO
,CONTRACTIBLE=COND

Specifies whether a multi-header cell pool is to be contractible. The default is CONTRACTIBLE=NO.

NO
The system is to use a multi-header cell pool that is not contractible.

COND
On a BUILD request, the system is to use a contractible multi-header cell pool when transaction execution is available or a non-contractible multi-header cell pool when transactional execution is not available. This parameter is valid only when MULTIHDR=YES is specified, and requires that CELLSPERCPU=1 be specified (meaning that only one cell is obtained per extent, not that there is only one total cell associated with each CPU).

On a GET or FREE request, the system is to use a contractible multi-header cell pool when transaction execution is available or a non-contractible multi-header cell pool when transactional execution is not available. The system performs this check at run time. This parameter is valid only when MULTIHDR=YES is specified.

Rule: If you specify CONTRACTIBLE=COND on the BUILD request for a cell pool, you must specify CONTRACTIBLE=COND on every GET or FREE request for that cell pool. To maintain performance efficiency, the system does not enforce this check.

,AUTOCONTRACT=YES
,AUTOCONTRACT=NO

On a BUILD request, specifies whether automatic contraction of the cell pool is to be initiated by the system based on other specified parameters. This parameter is valid only when CONTRACTIBLE=COND is specified. The default is AUTOCONTRACT=YES.

YES
The system will automatically initiate contraction based on the FREECELLS, FREECELLSDIVISOR, CONTINTVL, and FREECELLSPERCPU parameters. The contraction begins on a CPOOL FREE request. If there are no FREE requests, no contraction will occur.

NO
No automatic contraction will occur. Your program must explicitly initiate contraction, if desired, by issuing a CPOOL CONTRACT request.

,FREECELLS=*free-cells*
,FREECELLS=CPOOL_BUILD
On a BUILD request, specifies a fullword that identifies the target number of free cells at the conclusion of a contraction. When FREECELLSPERCPU is in effect, the maximum number allowed for FREECELLS is 128K (131072). For *free-cells*, you can specify a decimal number, an RS-type address of a fullword that contains the value, or a register (2) - (12) that contains the value. A value less than 0 is treated as 0. This parameter is required when you specify AUTOCONTRACT=YES.

The FREECELLS value undergoes the following calculation:

1. When FREECELLSPERCPU is not NO, the FREECELLS value is multiplied by the number of online CPUs that apply based on the FREECELLSPERCPU specification; otherwise, the FREECELLS value remains unchanged.
2. The total from step 1 is divided by the FREECELLSDIVISOR value, and the result is rounded down to a whole number.

On a CONTRACT request, you can specify a value of CPOOL_BUILD to indicate that you want to use the FREECELLS, FREECELLSDIVISOR, and FREECELLSPERCPU specifications that were provided on the CPOOL BUILD request.

,FREECELLSDIVISOR=*divisor*
On a BUILD request, specifies a fullword that identifies the divisor to be used in calculating the target number of free cells. For *divisor*, you can specify a decimal number, an RS-type address of a fullword that contains the value, or a register (2) - (12) that contains the value. A value less than 1 is treated as 1. This parameter is required when you specify AUTOCONTRACT=YES.

Use of FREECELLS with FREECELLSDIVISOR provides flexibility in case you do not want a whole number of free cells per applicable online CPU. For instance, a FREECELLS value of 3 and FREECELLSDIVISOR value of 2 would provide 1.5 free cells per applicable online CPU, with the total free cells target then rounded down to a whole number.

,CONTINTVL=*seconds*
On a BUILD request, a fullword that identifies the number of seconds that must elapse after the previous contraction (either automatically initiated or explicitly initiated) before the next automatic contraction may begin. The maximum value for CONTINTVL is the number of seconds in 365 days (31536000). For *seconds*, you can specify a decimal number, an RS-type address of a fullword that contains the value, or a register (2) - (12) that contains the value. A value less than 1 is treated as 1. This parameter is both required and valid only when you specify AUTOCONTRACT=YES.

,FREECELLSPERCPU=(*xxxx*,...,*zzzz*)
On a BUILD request, specifies the types of applicable online CPUs for calculating the target number of free cells. This parameter is required when you specify AUTOCONTRACT=YES. You must specify one (or more) of the following values for *xxxx* (through *zzzz*):

NO
No selection by CPU type will be done. The system uses the FREECELLS value you specify as the total.

When you omit NO, the target number of free cells will be calculated based on the number of applicable online CPUs.

ALL
All CPU types are applicable.

STANDARD
Standard CPs are applicable (not zAAPs; not zIIPs).

ZAAP
zAAPs are applicable.

ZIIP
zIIPs are applicable.

If you specify conflicting options, the last option takes effect. For instance, if you specify FREECELLSPERCPU=(ALL,NO), ALL would indicate that all CPU types are applicable, but then NO would indicate that no per-CPU processing is to be done, and that would be the result.

,PLISTVER=IMPLIED_VERSION
,PLISTVER=0
,PLISTVER=1

For CPOOL BUILD, an optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. The macro keys associated with each supported version of the macro are listed in macro usage note 3. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify the same value on all of the macro forms used for a request.

The values are:

IMPLIED_VERSION
The lowest version that allows all parameters specified on the request to be processed. If you omit the PLISTVER parameter, IMPLIED_VERSION is the default.

0
Supports all parameters except those specifically referenced in higher versions.

1
Supports the following parameters, in addition to those from version 0:

CONTRACTIBLE=COND
AUTOCONTRACT
FREECELLS
FREECELLSPERCPU
FREECELLDIVISOR
CONTINTVL

Use PLISTVER=1 only if your program is running on a system that supports z/OS V2R2 functions.

To code: Specify one of the following:

- `IMPLIED_VERSION`
- A decimal value of `0` or `1`.

ABEND codes

The CPOOL macro issues abend code X'C78'. See *z/OS MVS System Codes* for an explanation and possible responses.

Return codes

CPOOL BUILD, DELETE, FREE, and GET,UNCOND have no return codes. If any of these requests fail, CPOOL issues an abend.

CPOOL GET,COND returns a return code in register 1. See "Output register information" on page 258 for specific information.

CPOOL CONTRACT returns a hexadecimal return code in word 1 (bytes 4 through 7) of the work area used to return information to the calling program, as described in Table 42 on page 272.

Table 42. Return codes for the CPOOL CONTRACT service

Hexadecimal return code	Meaning and action
00	**Meaning**: The cell pool was defined to be contractible. **Action**: None.

Table 42. Return codes for the CPOOL CONTRACT service (continued)	
Hexadecimal return code	**Meaning and action**
04	**Meaning**: The cell pool was not defined to be contractible. **Action**: Do not issue a CPOOL CONTRACT request for a non-contractible cell pool. If you want the cell pool to be contractible, change the CPOOL BUILD request to specify the required parameters to build a contractible multi-header cell pool.

CPOOL LIST returns a hexadecimal return code in word 1 (bytes 4 through 7) of the work area used to return information to the calling program, as described in Table 43 on page 273.

Table 43. Return codes for the CPOOL LIST service	
Hexadecimal return code	**Meaning and action**
00	**Meaning**: Successful completion. **Action**: None.
01	**Meaning**: The work area holds all the information that fits 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.
02	**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.
03	**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 exists, ask the system programmer to request a dump to get more information for IBM support personnel.

Example 1

Create a cell pool containing 40-byte cells from subpool 2. Allow for 10 cells in the initial extent and 20 cells in all subsequent extents of the cell pool.

```
CPOOL BUILD,PCELLCT=10,SCELLCT=20,CSIZE=40,SP=2
```

Example 2

Create a cell pool containing 40-byte cells from subpool 231 (CSA). Allow for 10 cells in the initial extent and 20 cells in all subsequent extents of the cell pool. Indicate that the system is to assign the storage to the primary address space.

```
CPOOL BUILD,PCELLCT=10,SCELLCT=20,CSIZE=40,SP=231,OWNER=PRIMARY
```

Example 3

Unconditionally obtain a cell pool, specifying the pool ID in register 2. Use a PC instruction for linkage and do not save the registers.

```
CPOOL GET,U,CPID=(2),REGS=USE,LINKAGE=SYSTEM
```

Example 4

Free a cell specifying the pool ID in register 2 and the cell address in register 3.

```
CPOOL FREE,CPID=(2),CELL=(3)
```

Example 5

Delete a cell pool, specifying the pool ID in register 2. Use a PC instruction for linkage.

```
CPOOL DELETE,CPID=(2),LINKAGE=SYSTEM
```

Example 6

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 info 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
```

Example 7

Build a contractible cell pool, when available, with all contraction to be done by application use of CPOOL CONTRACT.

```
CPOOL  BUILD,MAXCELLS=(4),CELLSPERCPU=1,                      *
      CSIZE=(6),SP=(7),KEY=(8),TCB=(9),MULTIHDR=YES,          *
      CONTRACTIBLE=COND
```

Example 8

Build a contractible cell pool, when available, with automatic contraction every 120 seconds. The number of free cells is to be calculated as 3/2 (1.5) multiplied by the number of online standard CPs and zIIPs.

```
*Set up registers 4, 6, 7, 8, 9 for the corresponding keywords
CPOOL  BUILD,MAXCELLS=(4),CELLSPERCPU=1,                      *
      CSIZE=(6),SP=(7),KEY=(8),TCB=(9),MULTIHDR=YES,          *
      CONTRACTIBLE=COND,AUTOCONTRACT=YES,                     *
      FREECELLS=3,FREECELLSDIVISOR=2,                         *
      FREECELLSPERCPU=(STANDARD,ZIIP),CONTINTVL=120
*
*Set up registers 2, 3 for the corresponding keywords
```

```
CPOOL  GET,UNCOND,CPID=(2),CELL=(3),MULTIHDR=YES,             *
      CONTRACTIBLE=COND
*
*Set up registers 2, 3 for the corresponding keywords
CPOOL  GET,C,CPID=(2),CELL=(3),MULTIHDR=YES,                  *
      CONTRACTIBLE=COND
*
*Set up registers 2, 3 for the corresponding keywords
CPOOL  FREE,CPID=(2),CELL=(3),MULTIHDR=YES,                   *

CONTRACTIBLE=COND
*
*Set up registers 2, 3 for the corresponding keywords
CPOOL  CONTRACT,CPID=(2),FREECELLS=(3)
```

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.
␣	One or more blanks must precede CPOOL.
CPOOL	
␣	One or more blanks must follow CPOOL.
BUILD	
,PCELLCT=*primary cell count*	*cell count*: Symbol, decimal. **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, 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. **Default:** SP=0

Syntax	Description
,BNDRY=DWORD	**Default**: BNDRY=DWORD
,BNDRY=QWORD	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.
,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*	**Default:** The default depends on which subpool you specify. See the list of subpool characteristics in *z/OS MVS Programming: Authorized Assembler Services Guide* for information on storage keys for specific subpools.
,TCB=*tcb addr*	*tcb addr*: A-type address.
	Default: TCB address in PSATOLD.
,HDR=*hdr*	*hdr*: Character string enclosed in single quotation marks, A-type address.
,OWNER=HOME	**Default:** OWNER=HOME
,OWNER=PRIMARY	
,OWNER=SYSTEM	
,MULTIHDR=NO	**Default:** MULTIHDR=NO
,MULTIHDR=YES	
,MULTIHDR=COND	
,MAXCELLS=*mmmm*	
,CELLSPERCPU=*nnnn*	
,CELLSHARE=NO	**Default:** CELLSHARE=NO
,CELLSHARE=YES	

Syntax	Description
,CONTRACTIBLE=NO	**Default:** CONTRACTIBLE=NO
,CONTRACTIBLE=COND	
,AUTOCONTRACT=YES	**Default:** AUTOCONTRACT=YES
,AUTOCONTRACT=NO	
,FREECELLS=*free-cells*	*free-cells*: Decimal number, RS-type address, or register (2) - (12).
,FREECELLDIVISOR=*divisor*	*divisor*: Decimal number, RS-type address, or register (2) - (12).
,CONTINTVL=*seconds*	*seconds*: Decimal number, RS-type address, or register (2) - (12).
,FREECELLSPERCPU=(*xxxx*,...,*zzzz*)	*xxxx*,...,*zzzz*: One or more of: NO, ALL, STANDARD, ZAAP, ZIIP.
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=0	
,PLISTVER=1	
,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.
␣	One or more blanks must precede CPOOL.
CPOOL	
␣	One or more blanks must follow CPOOL.

Syntax	Description
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 format of the macro.
,SCELLCT=*secondary cell count*	**Default:** PCELLCT
,CSIZE=*cell size*	*cell size*: Symbol, decimal number, or register (0), (2) - (12). **Note:** CSIZE must be specified on either the list or the execute form of the macro.
,SP=*subpool number*	*subpool number*: Symbol, decimal number, or register (0), (2) - (12). **Default:** SP=0
,BNDRY=DWORD	**Default**: BNDRY=DWORD
,BNDRY=QWORD	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.
,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*	**Default:** The default depends on which subpool you specify. See the list of subpool characteristics in *z/OS MVS Programming: Authorized Assembler Services Guide* for information on storage keys for specific subpools.
,TCB=*tcb addr*	*tcb addr*: RX-type address or register (0), (2) - (12). **Default:** TCB address in PSATOLD.

Syntax	Description
,HDR=*hdr*	*hdr*: character string enclosed in single quotation marks, RX-type address, or register (0), (2) - (12).
,LINKAGE=SYSTEM	**Default:** LINKAGE=SYSTEM
,LINKAGE=BRANCH	
,OWNER=HOME	**Default:** OWNER=HOME
,OWNER=PRIMARY	
,OWNER=SYSTEM	
,MULTIHDR=NO	**Default:** MULTIHDR=NO
,MULTIHDR=YES	
,MULTIHDR=COND	
,MAXCELLS=*mmmm*	
,CELLSPERCPU=*nnnn*	
,CELLSHARE=NO	**Default:** CELLSHARE=NO
,CELLSHARE=YES	
,CONTRACTIBLE=NO	**Default:** CONTRACTIBLE=NO
,CONTRACTIBLE=COND	
,AUTOCONTRACT=YES	**Default:** AUTOCONTRACT=YES
,AUTOCONTRACT=NO	
,FREECELLS=*free-cells*	*free-cells*: Decimal number, RS-type address, or register (2) - (12).
,FREECELLDIVISOR=*divisor*	*divisor*: Decimal number, RS-type address, or register (2) - (12).
,CONTINTVL=*seconds*	*seconds*: Decimal number, RS-type address, or register (2) - (12).
,FREECELLSPERCPU=(*xxxx*,...,*zzzz*)	*xxxx*,...,*zzzz*: One or more of: NO, ALL, STANDARD, ZAAP, ZIIP.
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=0	
,PLISTVER=1	
,MF=(E,*list addr)*	*list addr*: 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,*list addr*)
Specifies the execute form of the CPOOL macro.

Chapter 34. 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 35. 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 44. 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 44. 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 44. 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 44. 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 44. 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 44. 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 36. CSVAPF — Control 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:

- Change the format of the APF list (from dynamic to static, and vice versa)
- Add or delete the library entries in a dynamic APF list (without having to reIPL the system)
- Determine whether or not a library is in the APF list
- Obtain a list of all library entries in the APF list
- Determine the current format (dynamic or static) of the APF list.

The CSVAPF macro is also described in the *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the REQUEST=ADD, REQUEST=DELETE, and REQUEST=DYNFORMAT parameters.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state and any PSW key. For the ADD, DELETE requests: RACF UPDATE authority to the FACILITY class entity CSVAPF.*libname*. For a DYNFORMAT request: RACF authority to the FACILITY class entity CSVAPF.MVS.SETPROG.FORMAT.DYNAMIC. If no RACF profile is defined or RACF is not available, one of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized
Dispatchable unit mode:	For the ADD, DELETE, and DYNFORMAT requests, task mode. For the QUERY, QUERYFORMAT, and LIST requests, task or SRB mode.
Cross memory mode:	For the ADD, DELETE, and DYNFORMAT requests, PASN = HASN = SASN. For the QUERY, QUERYFORMAT, and LIST requests, any PASN, any HASN, any SASN.
AMODE:	For a QUERY or QUERYFORMAT request, 31-bit. For all other requests, 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:	For the QUERY, QUERYFORMAT, and LIST requests, the local and CMS locks may be held, but are not required. For all other requests, no locks may be held.

Environmental factor	Requirement
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.
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=ADD	DSNAME, VOLTYPE, VOLUME, RETCODE, RSNCODE
REQUEST=DELETE	DSNAME, VOLTYPE, VOLUME, RETCODE, RSNCODE
REQUEST=DYNFORMAT	RETCODE, RSNCODE
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).

Syntax	Description
,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=ADD
REQUEST=DELETE
REQUEST=LIST
REQUEST=DYNFORMAT

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.

ADD
Add a library to the dynamic APF list. To use this parameter, the format of the APF list must be dynamic.

DELETE
Delete a library from the dynamic APF list. To use this parameter, the format of the APF list must be dynamic.

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.

DYNFORMAT
Change the format of the APF list from static to dynamic. Before you make the change, contact the system programmer to validate that all programs and vendor products are converted to use dynamic APF services and that the proper program products are installed.

,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:

1. 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.
2. Defining only the alias data set does not authorize either the real or the alias data set. A real data set name must be specified.

,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.

,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 45. 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. • ADD - The system added the specified library to the APF list. • DELETE - The system deleted the specified library from the APF list. • LIST - The system returned a list of all the libraries in the APF list. • DYNFORMAT - The format of the APF list is changed (from static to dynamic). **Action**: None.
04	xxxx0401	**Meaning**: The CSVAPF request completed successfully. The result depends on the option: • For a QUERY request, the library is in the APF list, and is SMS-managed. • For an ADD request, the library is already in the APF list. **Action**: None.
04	xxxx0402	**Meaning**: One of the following: • For a QUERY request, the library is not in the APF list • For a DELETE 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.

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

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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	xxxx0802	**Meaning**: Program error. A program running in SRB mode entered a request that required task mode. **Action**: For the specified request, avoid issuing the CSVAPF macro while running in SRB mode.
08	xxxx0803	**Meaning**: Program error. A program issued the CSVAPF macro while running disabled for I/O and external interrupts. **Action**: Issue the CSVAPF macro while running enabled for I/O and external interrupts.
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 "Environment" on page 307 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.
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.

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

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
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.
08	xxxx080F	**Meaning**: Program error. For an ADD, DELETE, or DYNFORMAT request, the caller holds a lock. **Action**: Release any held locks before issuing CSVAPF with the specified request.
0C	xxxx0C01	**Meaning**: Environmental error. The function is not available. The APF list format is static. **Action**: If desired, issue the CSVAPF macro with the REQUEST=DYNFORMAT parameter to change the format of the APF list to dynamic (contact the system programmer to ensure that all the required software products are updated and all vendor products are converted). Then try the function again.
0C	xxxx0C02	**Meaning**: Environmental error. The function is not available. DFSMS/MVS 1.1 is not installed. **Action**: Contact the system programmer. Provide the return code, the reason code, and the explanation of the error.

Table 45. Return and Reason Codes for the CSVAPF Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
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

Add SMS-managed library MY.LIBRARY.NAME to the list of APF-authorized libraries:

```
         CSVAPF REQUEST=ADD,DSNAME=MYLIB,VOLTYPE=SMS,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
   .
   .
MYLIB    DC    CL44'MY.LIBRARY.NAME'
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
         CSVAPFAA ,            Include CSVAPFAA mapping macro
```

Example 2

Add library MY.LIBRARY.NAME on volume 861234 to the list of APF=authorized libraries,

```
         CSVAPF REQUEST=ADD,DSNAME=MYLIB,VOLUME=MYVOL,VOLTYPE=ANY,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
   .
   .
MYLIB    DC    CL44'MY.LIBRARY.NAME'
MYVOL    DC    CL6'861234'
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
         CSVAPFAA ,            Include CSVAPFAA mapping macro
```

Example 3

Change the format of the APF list from static to dynamic:

```
         CSVAPF REQUEST=DYNFORMAT,RETCODE=LRETCODE,RSNCODE=LRSNCODE
   .
   .
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
         CSVAPFAA ,            Include CSVAPFAA mapping
```

Example 4

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 5

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.
	Valid parameters (Required parameters are underlined):

Syntax	Description
REQUEST=QUERY	DSNAME, VOLTYPE, VOLUME, RETCODE, RSNCODE
REQUEST=ADD	DSNAME, VOLTYPE, VOLUME, RETCODE, RSNCODE
REQUEST=DELETE	DSNAME, VOLTYPE, VOLUME, RETCODE, RSNCODE
REQUEST=DYNFORMAT	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 37. CSVDYLPA — Provide dynamic LPA services

Description

CSVDYLPA allows you to request dynamic LPA services. Be aware, however, that changes to LPA itself are not actually done. This set of services truly only lets you add modules to, and delete modules from, common storage. When searching by module name, the system will locate the copy of a module added by dynamic LPA services even if it was present in PLPA, MLPA, or FLPA.

With CSVDYLPA, you can request services to:

- Add one or more modules to common storage (REQUEST=ADD).
- Delete one or more modules that were previously added using dynamic LPA services (REQUEST=DELETE).
- Request that your program wait until processing of LPA statements in the IPL-time PROGxx parmlib members complete (REQUEST=DEFLPAWAIT).
- Query information about support for LPA services (REQUEST=QUERYDYN).
- Query whether processing of LPA statements in the IPL-time PROGxx parmlib members is complete (REQUEST=QUERYDEFLPA).

Following the descriptions of the requests are:

- The return and reason codes, "Return and reason codes" on page 319
- Examples of using CSVDYLPA, "Example 1" on page 324

Return and reason codes

When the CSVDYLPA 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 CSVLPRET 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 46. Return and Reason Codes for the CSVDYLPA Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: CsvdylpaRc_OK **Meaning**: CSVDYLPA request successful. **Action**: None required. **ADD** **Meaning**: All modules successfully added to LPA. **Action**: None required. **DELETE** **Meaning**: All modules removed from LPA. **Action**: None required. **DEFLPAWAIT** **Meaning**: Deferred LPA processing has completed. **Action**: The application may use LPA modules that were to be added by deferred LPA processing.
4	—	**Equate Symbol**: CsvdylpaRc_Warn **Meaning**: Warning **Action**: Refer to the action provided with the specific reason code.
4	xxxx0401	**Equate Symbol**: CsvdylpaRsnNotAllSuccessful **Meaning**: For ADD and DELETE request, at least one input module could not be processed successfully. Information about the problem is contained within the MODINFO entry for that module, in field LpmeaOutputFlags (for ADD) or field LpmedOutputFlags (for DELETE). The system continued to process entries after the one for which the problem occurred. **Action**: Fix the problem before requesting the function again.
8	—	**Equate Symbol**: CsvdylpaRc_InvParm **Meaning**: CSVDYLPA request specifies parameters that are not valid. For ADD and DELETE, when the problem occurred while processing a particular MODINFO entry, the system will not process any additional MODINFO entries. **Action**: Refer to the action provided with the specific reason code.
8	xxxx0801	**Equate Symbol**: CsvdylpaRsnBadParmlist **Meaning**: Unable to access parameter list. **Action**: Check for possible storage overlay.
8	xxxx0802	**Equate Symbol**: CsvdylpaRsnSrbMode **Meaning**: SRB mode. **Action**: Avoid requesting this function in SRB mode.
8	xxxx0803	**Equate Symbol**: CsvdylpaRsnNotEnabled **Meaning**: Not Enabled. **Action**: Avoid requesting this function while not enabled.
8	xxxx0805	**Equate Symbol**: CsvdylpaRsnHomeNotPrimary **Meaning**: Home address space different from primary address space. **Action**: Avoid requesting this function in this environment.
8	xxxx0809	**Equate Symbol**: CsvdylpaRsnBadRequestType **Meaning**: Request type is not valid. **Action**: Check for possible storage overlay of the parameter list.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx080A	**Equate Symbol**: CsvdylpaRsnBadEstaex **Meaning**: Unable to establish ESTAEX. "xxxx" contains the ESTAEX return code. There could be an FRR established. **Action**: Refer to documentation for ESTAEX return code "xxxx".
8	xxxx080B	**Equate Symbol**: CsvdylpaRsnReservedNot0 **Meaning**: Reserved field not 0. **Action**: Check for possible storage overlay of the parameter list.
8	xxxx080D	**Equate Symbol**: CsvdylpaRsnBadParmlistALET **Meaning**: Unable to use ALET of parameter list. **Action**: Make sure that the ALET of the parameter list is valid. The access register might not have been set up correctly.
8	xxxx080E	**Equate Symbol**: CsvdylpaRsnBadVersion **Meaning**: Bad version number. **Action**: Check for possible storage overlay of the parameter list.
8	xxxx080F	**Equate Symbol**: CsvdylpaRsnLocked **Meaning**: Locked **Action**: Avoid requesting this function in this environment.
8	xxxx0815	**Equate Symbol**: CsvdylpaRsnBadDsnameArea **Meaning**: Unable to access data set name. **Action**: Make sure that the DSNAME area is valid.
8	xxxx0816	**Equate Symbol**: CsvdylpaRsnBadModinfoArea **Meaning**: Unable to access MODINFO area. **Action**: Make sure that the MODINFO area is valid.
8	xxxx0817	**Equate Symbol**: CsvdylpaRsnBadModinfoALET **Meaning**: Unable to use ALET of MODINFO area. **Action**: Make sure that the ALET of the MODINFO area is valid. The access register might not have been set up correctly.
8	xxxx0818	**Equate Symbol**: CsvdylpaRsnBadOpen **Meaning**: Unable to open specified data set. **Action**: Make sure that you specified the proper data set, that it is a PDS or PDSE program library, and that it can be located by the system.
8	xxxx081D	**Equate Symbol**: CsvdylpaRsnBadNumMod **Meaning**: The value provided by the NUMMOD parameter is 0 or exceeds 256. **Action**: Specify a non-zero NUMMOD parameter value. Instead of providing more than 256 entries in a single call, use multiple calls each of which provides no more than 256 entries.
8	xxxx0820	**Equate Symbol**: CsvdylpaRsnBadDsnameALET **Meaning**: Bad dsname ALET. **Action**: Make sure that the ALET of the DSNAME area is valid. The access register might not have been set up correctly.
8	xxxx0822	**Equate Symbol**: CsvdylpaRsnBadModuleName **Meaning**: Bad modulename - first character is 0 or blank. **Action**: Provide a valid module name.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0823	**Equate Symbol**: CsvdylpaRsnBadDsname **Meaning**: Bad DSNAME - first character is 0 or blank. **Action**: Provide a valid data set name.
8	xxxx0829	**Equate Symbol**: CsvdylpaRsnBadAlloc **Meaning**: Unable to allocate data set. **Action**: Make sure that you specified the proper data set, that it is a PDS or PDSE program library, and that it can be located by the system.
8	xxxx082B	**Equate Symbol**: CsvdylpaRsnFunctionNotAvailable **Meaning**: Required DFSMS function or dynamic allocation is not available. **Action**: Make sure that the required DFSMS support is installed. Avoid requesting the function in an environment where dynamic allocation is not available.
8	xxxx082C	**Equate Symbol**: CsvdylpaRsnNotAuthDCB **Meaning**: Not authorized to use DCB option. Must be supervisor state, PKM allowing key 0-7, PSW key 0-7, or APF authorized. **Action**: Avoid using the DCB option unless you have the required authorization.
8	xxxx082D	**Equate Symbol**: CsvdylpaRsnNotAuthConcat **Meaning**: If not supervisor state, PKM allowing key 0-7, PSW key 0-7, or APF authorized, or if APFREQUIRED=YES is specified or defaulted, the concatenation represented by the input DDNAME or DCB must be APF authorized. **Action**: Avoid using a non-APF authorized concatenation unless you have the required authorization.
8	xxxx082E	**Equate Symbol**: CsvdylpaRsnNotAuthMemberMask **Meaning**: Not authorized to use MemberMask option. Must be supervisor state, PKM allowing key 0-7, PSW key 0-7, or APF authorized. **Action**: Avoid using the MODINFOTYPE=MEMBERMASK function unless you have the required authorization.
8	xxxx0830	**Equate Symbol**: CsvdylpaRsnBadModinfoxArea **Meaning**: Unable to access MODINFOX area. **Action**: Make sure that the MODINFOX area is valid.
8	xxxx0831	**Equate Symbol**: CsvdylpaRsnBadModinfoxALET **Meaning**: Unable to use ALET of MODINFOX area. **Action**: Make sure that the ALET of the MODINFOX area is valid. The access register might not have been set up correctly.
8	xxxx0833	**Equate Symbol**: CsvdylpaRsnNotESVC **Meaning**: An extended SVC was selected, but the specified SVC number is not an extended SVC. **Action**: Correct the SVC number.
8	xxxx0834	**Equate Symbol**: CsvdylpaRsnBadESvcrnum **Meaning**: The routing number for the selected extended SVC exceeded the number of entries for that extended SVC that were defined at IPL **Action**: Correct the extended SVC routine number.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx083C	**Equate Symbol**: CsvdylpaRsnNotPartitioned **Meaning**: For ADD request, the data set is not partitioned. **Action**: Make sure that you specified the proper data set and that it is a PDS or PDSE program library.
8	xxxx083D	**Equate Symbol**: CsvdylpaRsnBadByaddrInfo **Meaning**: For ADD request with BYADDR=YES, the module information is incorrect. **Action**: Make sure that the entry point and load point addresses represent common area storage. Make sure that the entry point lies within the primary load segment.
8	xxxx083E	**Equate Symbol**: CsvdylpaRsnNotAuthByaddr **Meaning**: Not authorized to use BYADDR=YES option. Must be supervisor state, PKM allowing key 0-7, PSW key 0-7, or APF authorized. **Action**: Avoid using BYADDR=YES unless you have the required authorization.
8	xxxx083F	**Equate Symbol**: CsvdylpaRsnBadDcbArea **Meaning**: Unable to access the opened DCB. **Action**: Make sure that the DCB has been opened.
8	xxxx0840	**Equate Symbol**: CsvdylpaRsnEnqHeldShared **Meaning**: The ENQ resource with QNAME SYSZCSV and RNAME CSVDYLPA was held in the shared state on entry to dynamic LPA services. **Action**: Avoid holding the ENQ shared when using dynamic LPA services.
8	xxxx0841	**Equate Symbol**: CsvdylpaRsnBadLPMEAQArea **Meaning**: Unable to access LPMEAQ area. **Action**: Make sure that the LPMEAQ area is valid.
8	xxxx0842	**Equate Symbol**: CsvdylpaRsnBadLPMEAQALET **Meaning**: Unable to use ALET of LPMEAQ area. **Action**: Make sure that the ALET of the LPMEAQ area is valid. The access register might not have been set up correctly.
8	xxxx0843	**Equate Symbol**: CsvdylpaRsnNotAuthAddAlias **Meaning**: Not authorized to use the ADDALIAS=YES function. Must be supervisor state, PKM allowing key 0-7, PSW key 0-7, or APF authorized. **Action**: Do not use the ADDALIAS=YES function unless you have the required authorization.
8	xxxx0844	**Equate Symbol**: CsvdylpaRsnBadPathnameLen **Meaning**: The PATHNAMELEN parameter value is not in the range 1-1023. **Action**: Provide a valid PATHNAMELEN parameter value.
8	xxxx0845	**Equate Symbol**: CsvdylpaRsnBadPathnameArea **Meaning**: Unable to access the path name. **Action**: Make sure that the PATHNAME area is valid.
8	xxxx0846	**Equate Symbol**: CsvdylpaRsnBadPathnameALET **Meaning**: Unable to use ALET of PATHNAME area. **Action**: Make sure that the ALET of the PATHNAME area is valid. The access register might not have been set up correctly.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0847	**Equate Symbol**: CsvdylpaRsnBadPathnameNumMod **Meaning**: PATHNAME was specified and the value provided by the NUMMOD parameter is not 1. **Action**: Provide only one entry for each call.
8	xxxx0848	**Equate Symbol**: CsvdylpaRsnNotAuthDEFLPAWAIT **Meaning**: Not authorized to use REQUEST=DEFLPAWAIT. Must be supervisor state, PKM allowing key 0-7, PSW key 0-7, or APF authorized. **Action**: Do not use the REQUEST=DEFLPAWAIT function unless you have the required authorization.
C	—	**Equate Symbol**: CsvdylpaRc_Env **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	xxxx0C02	**Equate Symbol**: CsvdylpaRsnNoStorage **Meaning**: There is not sufficient storage to complete the request. **Action**: Contact your system programmer. There is a shortage of common storage.
C	xxxx0C04	**Equate Symbol**: CsvdylpaRsnBadDirectory **Meaning**: When using the MemberMask option, the data set directory was in error. Either an I/O error occurred accessing the directory, or the format of a directory entry was incorrect. **Action**: Fix the data set directory. Make sure that the data set is a PDS or PDSE program library.
C	xxxx0C05	**Equate Symbol**: CsvdylpaRsnStoragelimExceeded **Meaning**: For ADD request, the amount of module storage needed for the request would have caused the amount of CSA or ECSA remaining to fall below the threshold specified by the system programmer using the LPA CSAMIN statement of PROGxx, the SETPROG LPA,CSAMIN system command, or CSA/ECSA specified in IEASYSxx. **Action**: Specify that fewer modules be added, or have the system programmer reduce the CSAMIN amounts.
10	—	**Equate Symbol**: CsvdylpaRC_CompError **Meaning**: Unexpected failure. **Action**: Refer to the action provided with the specific reason code.
10	xxxx1001	**Equate Symbol**: CsvdylpaRsnCompError **Meaning**: Unexpected failure. The state of the request is unpredictable. **Action**: Contact your system programmer.

Examples

Example 1

Operation 1:

1. Add a module to LPA
2. Delete a module from LPA

The code is as follows:

```
*************************************************************
* Set up MODINFO area, for module "MYMODULE", indicating    *
* that the module is to be page-fixed.                      *
* Add the module to LPA, locating the module using          *
* data set 'SYS1.MYDS'.                                     *
*************************************************************
         LA    2,ADDINFO
         USING LPMEA,2
         XC    ADDINFO(LPMEA_LEN),ADDINFO
         MVC   LPMEANAME,=CL8'MYMODULE'
         OI    LPMEAINPUTFLAGS0,LPMEAFIXED
         DROP  2
         CSVDYLPA REQUEST=ADD,MODINFOTYPE=MEMBERLIST,        *
               MODINFO=(2),NUMMOD=LMODN,                     *
               DSNAME=LDS1,                                  *
               REQUESTOR=LREQ,                               *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,DYLPAL)
*
* Place code to check return/reason codes here
*
 ... processing code here ...
*
*************************************************************
* Set up MODINFO area, for module "MYMODULE", using the     *
* token returned on REQUEST=ADD.                            *
* Delete the module from LPA.                               *
*************************************************************
         LA    3,ADDINFO
         USING LPMEA,3
         LA    2,DELINFO
         USING LPMED,2
         XC    DELINFO(LPMED_LEN),DELINFO
         MVC   LPMEDNAME,LPMEANAME
         MVC   LPMEDDELETETOKEN,LPMEADELETETOKEN
         DROP  2,3
         CSVDYLPA REQUEST=DELETE,MODINFO=(2),NUMMOD=LMODN,   *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,            *
               MF=(E,DYLPAL)
*
* Place code to check return/reason codes here.
*
LMODN    DC    F'1'
LDS1     DC    CL44'SYS1.MYDS'
LREQ     DC    CL16'CSVDYLPA EXAMPLE'
         CSVLPRET              Return code information
DYNAREA  DSECT
ADDINFO  DS    0D
         ORG   ADDINFO+LPMEA_LEN
DELINFO  DS    0D
         ORG   DELINFO+LPMED_LEN
LRETCODE DS    F
LRSNCODE DS    F
         CSVDYLPA MF=(L,DYLPAL)
```

Example 2

Operation 2:

1. Determine if deferred LPA processing is complete.
2. Wait for the completion of deferred LPA processing, if needed.

The code is as follows:

```
CSVDYLPA REQUEST=QUERYDEFLPA,                              *
      DEFLPASTATE=LDEFLPASTATE
CLI   LDEFLPASTATE,CsvdylpaDefLPAState_Complete
JE    DEFLPA_COMPLETE
CSVDYLPA REQUEST=DEFLPAWAIT,                               *
      RETCODE=LRETCODE,RSNCODE=LRSNCODE,                   *
      MF=(E,DYLPAL)
*
* Place code to check return/reason codes here
*
DEFLPA_COMPLETE DS 0H
 ... processing code here ...
DYNAREA  DSECT
```

```
LRETCODE DS    F
LRSNCODE DS    F
         CSVDYLPA MF=(L,DYLPAL)
LDEFLPASTATE DS X
         CSVLPRET
```

REQUEST=ADD option of CSVDYLPA

REQUEST=ADD allows you to add one or more modules or aliases to LPA.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Any of the following: • Supervisor state • PKM 0-7 • PSW key 0-7 • APF-authorized • In addition to any of the above, if SECMODCHECK=YES is specified or defaulted to, issuer must be authorized for UPDATE to the RACF FACILITY class resource CSVDYLPA.ADD.modname Users of MODINFOTYPE=MEMBERMASK and ADDALIAS=YES require any of the following: • Supervisor state • PKM 0-7 • PSW key 0-7 • APF-authorized Users of DCB, DCBPTR, MASKDCB, or MASKDCBPTR require any of the following: • Supervisor state • PKM 0-7 • PSW key 0-7 • APF-authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.

Environmental factor	Requirement
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided data set name via the DSNAME parameter has the same requirements and restrictions as the control parameters. The user-provided information via the MODINFO parameter has the same requirements and restrictions as the control parameters. The user-provided information through the PATHNAME parameter has the same requirements and restrictions as the control parameters.

Programming requirements

The caller should include the CSVLPRET macro to get equate symbols for the return and reason codes.

The caller must include the CSVLPRET macro to get a mapping of the input/output area provided via the MODINFO

The caller may hold the system ENQ resource with QNAME SYSZCSV and RNAME CSVDYLPA in the exclusive state. While this ENQ resource is held, any other requests to use the CSVDYLPA services to ADD or DELETE will be delayed. The ENQ resource must not be held in the shared state when using CSVDYLPA services parameter.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

Before issuing the CSVDYLPA macro, the caller does not need to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CSVDYLPA macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYLPA.
CSVDYLPA	
ƀ	One or more blanks must follow CSVDYLPA.
REQUEST=ADD	
,MODINFOTYPE=MEMBERLIST	
,MODINFOTYPE=MEMBERMASK	
,MODINFO=*modinfo*	*modinfo:* RS-type address or address in register (2) - (12)
,NUMMOD=*nummod*	*nummod:* RS-type address or address in register (2) - (12)
,BYADDR=NO | YES	**Default:** BYADDR=NO
,BYPATH=NO | YES	**Default:** BYPATH=NO
,PATHNAMELEN=*pathnamelen*	*pathnamelen*: RS-type address or address in register (2) - (12)

Syntax	Description
,PATHNAME=*pathname*	*pathname*: RS-type address or address in register (2) - (12)
,UCBADDR=NO_UCBADDR	**Default:** UCBADDR=NO_UCBADDR
,UCBADDR=*ucbaddr*	*ucbaddr*: RS-type address or address in register (2) - (12)
,CCHH=NO_CCHH	**Default:** CCHH=NO_CCHH
,CCHH=*cchh*	*cchh*: RS-type address or address in register (2) - (12)
,DSNAME=*dsname*	*dsname*: RS-type address or address in register (2) - (12)
,VOLUME={volume | CATALOG}	*volume* RS-type address or address in register (2) - (12)
,DDNAME=*ddname*	*ddname:* RS-type address or address in register (2) - (12)
,DCB=*dcb*	*dcb:* RS-type address or address in register (2) - (12)
,DCBPTR=*dcbptr*	*dcbptr:* RS-type address or address in register (2) - (12)
,PATHNAME=*pathname*	*pathname*: RS-type address or address in register (2) - (12)
,MODINFO=*modinfo*	*modinfo:* RS-type address or address in register (2) - (12)
,MODINFOX=*modinfox*	*modinfox:* RS-type address or address in register (2) - (12)
,MODINFOX=NO_MODINFOX	**Default:** MODINFOX=NO_MODINFOX
,MASKDSNAME=*maskdsname*	*maskdsname:* RS-type address or address in register (2) - (12)
,MASKDDNAME=*maskddname*	*maskddname:* RS-type address or address in register (2) - (12)
,MASKDCB=*maskdcb*	*maskdcb:* RS-type address or address in register (2) - (12)
,MASKDCBPTR=*maskdcbptr*	*maskdcbptr:* RS-type address or address in register (2) - (12)
,OUTAREAPTR=*outareaptr*	*outareaptr:* RS-type address or address in register (2) - (12)
,OUTAREALEN=*outarealen*	*outarealen:* RS-type address or address in register (2) - (12)
,OUTAREANUM=*outareanum*	*outareanum:* RS-type address or address in register (2) - (12)
,OUTAREASP=*outareasp*	*outareasp:* RS-type address or address in register (2) - (12)
,APFREQUIRED=YES | NO	**Default:** APFREQUIRED=YES

Syntax	Description
,SECMODCHECK=YES | NO	**Default:** SECMODCHECK=YES
,REQUESTOR=*requestor*	*requestor:* RS-type address or address in register (2) - (12)
,MODPROB=CONTINUE	**Default:** MODPROB=CONTINUE
,MODPROB=STOP	
,ERRORDATA=*errordata*	*errordata:* RS-type address or address in register (2) - (12)
,ADDALIAS=NO | YES	**Default:** ADDALIAS=NO
,QUERYONLY=NO | YES	**Default:** QUERYONLY=NO
,AC1=YES | NO	**Default:** AC1=YES
,LONGPARMOK=NO | YES	**Default:** LONGPARMOK=NO
,LPMEAQ=lpmeaq	*lpmeaq:* RS-type address or register (2) - (12).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

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

REQUEST=ADD
A required parameter. REQUEST=ADD indicates to add one or more modules to LPA. Note that if the module already exists within LPA, the request is still processed. The original copy of the module is not deleted. A new copy of the module is created, and subsequent searches of LPA will only find that new copy. A token is returned for each load module. The token must be used on the DELETE request unless using the TYPE=CURRENT or TYPE=OLDEST option of REQUEST=DELETE. This token is in the LPMEA area.

Modules added to the system via dynamic LPA processing are placed into CSA or ECSA storage. Therefore, it is important to ensure that the system CSA and ECSA sizes are adequately defined to handle the additional consumption of CSA storage resulting from the issuance of the dynamic LPA request.

,MODINFOTYPE=MEMBERLIST
,MODINFOTYPE=MEMBERMASK
A required parameter that indicates the type of MODINFO area provided. In all cases, the MODINFO area contains entries mapped by DSECT LPMEA in macro CSVLPRET. Each entry contains a flag area which should be cleared before calling the CSVDYLPA service.

You can indicate:

- The module is to be placed into fixed storage (as opposed to pageable),
- Only the whole pages within the module are to be placed into page-protected storage,
- The storage acquired for the module is to be OWNER=SYSTEM (as opposed to OWNER=HOME).

,MODINFOTYPE=MEMBERLIST
Indicates that the area contains the list of modules to be processed.

,MODINFOTYPE=MEMBERMASK
Indicates that the area is input/output, and contains only one entry. The module name in that entry is treated as a "mask". All members in the data set represented by the input (data set, DCB, or DDNAME) that match the input mask (the mask can contain wildcard characters "*" and "?" and a wildcard match is done) are processed. The system first creates a list of the members that are to be processed and then continues in the same manner as for MODINFOTYPE=MEMBERLIST.

,MODINFO=*modinfo*
When MODINFOTYPE=MEMBERLIST is specified, a required input/output parameter that specifies an area that contains contiguous entries. Each entry contains a module (or alias) name and status flags. Each name in the area must be unique. If a module has aliases, the module name and all associated aliases must be specified. The system processes more efficiently if the names in the area are in ascending EBCDIC order (e.g., "A", then "B", then "C"). The number of entries must match the value provided through the nummod parameter.

On output, among other possibilities, the status area might indicate:

- The module could not be located.
- An error occurred processing the input data set (in which case no further processing of additional modules is done).
- You are not authorized to process the particular module.

For a complete list of the possible problem types, refer to the equate symbols beginning with LpmeaModprob in the CSVLPRET data area.

If you specify an alias for an existing load module, the processing will not simply add that alias. Rather, it will create a new copy of the load module that is associated with that alias.

Within each LPMEA, the module name should be left-align padded on the right with EBCDIC blanks if less than 8 characters long. There should be no embedded blanks.

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

,NUMMOD=*nummod*
When MODINFOTYPE=MEMBERLIST is specified, a required input parameter that contains the number of entries in the area specified by the MODINFO parameter and in the area specified by the MODINFOX parameter. If PATHNAME is specified, NUMMOD must be one.If PATHNAME is not specified, NUMMOD must be in the range 1-256.

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

,BYADDR=NO | YES
When MODINFOTYPE=MEMBERLIST is specified, an optional parameter that indicates whether the modules have already been fetched into storage. The default is BYADDR=NO.

,BYADDR=NO
Indicates that the modules need to be fetched.

,BYADDR=YES
Indicates that the modules have already been fetched and are linkedited as authorized (with AC=1). In this case, the MODINFO area must be present not only with the module names but also with the following LPMEA fields set for each module:

- LpmeaEntryPointAddr - the entry point address (with bit 0 of that address being on if the entry is to receive control in AMODE 31).
- LpmeaLoadPointAddr - the load point address (the start) of the primary segment.
- LpmeaModlen - the length of that load segment.
- LpmeaLoadPointAddr2 - if the load module represents a split RMODE load module, the load point address of the secondary segment; otherwise 0.
- LpmeaModlen2 - if the load module represents a split RMODE load module, the length of that secondary load segment; otherwise 0.

All addresses must be in common storage, and the entry point address must be within the primary segment. It is up to the caller to page fix and page protect the storage as needed.

,BYPATH=NO | YES
When BYADDR=YES and MODINFOTYPE=MEMBERLIST are specified, an optional input parameter that indicates whether the module was fetched from a z/OS UNIX executable file that is specified as a fully qualified path name. The default is BYPATH=NO.

,BYPATH=NO
Indicates that the modules were not fetched from a z/OS UNIX executable file specified as a fully qualified path name.

,BYPATH=YES
Indicates that the module is fetched from a z/OS UNIX executable file specified as a fully qualified path name.

Note: The NUMMOD parameter must specify a value of one when using this option.

,UCBADDR=NO_UCBADDR
,UCBADDR=*ucbaddr*
When BYPATH=NO, BYADDR=YES and MODINFOTYPE=MEMBERLIST are specified, an optional input parameter that specifies the UCB address for the volume on which the first extent of the data set that contains the module exists. The UCB address is found from the DEBUCBA field. The DEBUCBA field is mapped by IEZDEB in the DEB associated with the open DCB that is used to load the module. You can use the UCB in either of the following ways:

- Pass the address from the DEBUCBA field into the IOSCAPF service, and use the output from the IOSCAPF service for the CSVDYLPA UCBADDR parameter.
- Use the address from the DEBUCBA field directly.

Default: NO_UCBADDR

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

,CCHH=NO_CCHH
,CCHH=*cchh*
When UCBADDR=*ucbaddr*, BYPATH=NO, BYADDR=YES and MODINFOTYPE=MEMBERLIST are specified, a required input parameter that contains the CC and HH values associated with the first extent of the data set that contains the module. The CCHH value comes from the DEBSTRCC and DEBSTRHH fields. These two fields are mapped by IEZDEB in the DEB associated with the open DCB that is used to load the module. The DEB entry corresponds to the UCB address that is provided by the UCBADDR parameter.

The CCHH parameter denotes a 32–bit track address consisting of a cylinder number and a track number. The exact number of bits that represent the cylinder number and track number is device dependent. The format of the fields is not relevant to the specification of the CCHH parameter.

Default: NO_CCHH

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

PATHNAMELEN=*pathnamelen*
When PATHNAME=*pathname*, BYADDR=NO and MODINFOTYPE=MEMBERLIST are specified, a required input parameter that is the length of the path name provided by the PATHNAME parameter.

Value range: 1 to 1023

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

PATHNAMELEN=*pathnamelen*
When BYPATH=YES, BYADDR=YES and MODINFOTYPE=MEMBERLIST are specified, a required input parameter that is the length of the path name provided by the PATHNAME parameter.

Value range: 1 to 1023

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

PATHNAME=*pathname*
When BYPATH=YES, BYADDR=YES and MODINFOTYPE=MEMBERLIST are specified, a required input parameter that is the fully qualified path name of the file from which the module was fetched. It cannot be a relative path name. The length of the path name should be in the range of 1 to 1023.

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

,DSNAME=*dsname*
,DDNAME=*ddname*
,DCB=*dcb*
,DCBPTR=*dcbptr*
,PATHNAME=*pathname*
When BYADDR=NO and MODINFOTYPE=MEMBERLIST are specified, a required input parameter.

,DSNAME=*dsname*
A parameter that contains the name of the data set/library from which all the input modules are to be loaded. The data set must be cataloged. It may be allocated as a PDS or PDSE program library.

Note that if the data set is migrated, the issuer's unit of work will wait until the data set is retrieved before continuing.

If the caller is authorized only by the RACF FACILITY class resource (is not supervisor state, system key, system PKM, or APF-authorized), then the data set must be APF-authorized. Similarly, if the caller is supervisor state, system key, or system PKM, or APF-authorized, the data set need not be APF-authorized.

To code: Specify the RS-type address, or address in register (2)-(12), of a 44-character field.

,VOLUME=*volume*
When DSNAME=dsname is specified, an optional parameter that contains the 6-character volume ID on which the data set resides.

Default: CATALOG which indicates to use the volume ID in the catalog.

To code: Specify the RS-type address, or address in register (2)-(12) of a 6-character field.

,DDNAME=*ddname*
A parameter that contains the DDNAME of the data set/library (or concatenation of libraries) from which all the input modules are to be loaded. The system will open the DDNAME for input.

If the caller is authorized only by the RACF FACILITY class resource (is not supervisor state, system key, system PKM, or APF-authorized), then the concatenation must be APF-authorized.

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

,DCB=*dcb*
A parameter that contains the opened DCB representing the data set/library (or concatenation of libraries) from which all the input modules are to be loaded. The DCB must be opened for input.

The caller must be either supervisor state, system key, system PKM, or APF-authorized to use this option.

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

,DCBPTR=*dcbptr*
A parameter that contains the address of the opened DCB representing the data set/library (or concatenation of libraries) from which all the input modules are to be loaded. The DCB must be opened for input. You can specify DCBPTR=CVTLINK to request that the LNKLST be used as the library concatenation (you must have an assembler USING established on the CVT data area in order to use this).

The caller must be either supervisor state, system key, system PKM, or APF-authorized to use this option.

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

PATHNAME=*pathname*
A parameter that is the file name from which the module is to be fetched. This parameter must be a fully qualified path name, which includes the file name. This parameter cannot be a relative path name. The file must be marked as an authorized program.

Note: NUMMOD must be one.

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

,MODINFO=*modinfo*
When MODINFOTYPE=MEMBERMASK is specified, a required input/output parameter that specifies an area that contains a single entry. The entry contains a member name mask and status flags.

Within the LPMEA, the module name field represents a member name mask. All members that match this mask (using wildcard matching with "*" representing 0 or more characters, and "?" representing exactly one character) will be processed. The member name mask should be left-align within the 8-byte field and padded on the right with EBCDIC blanks if less than 8 characters long. There should be no embedded blanks.

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

,MODINFOX=*modinfox*
When MODINFOTYPE=MEMBERLIST is specified, an optional input parameter that specifies an area that contains contiguous entries. Each entry contains additional information that is used when processing the information in the corresponding entry in the MODINFO area. MODINFOX is mapped by DSECT LPMEAX in macro CSVLPRET.

For a given entry name, you can specify that an SVC table entry be updated by the system. To do this, you must specify the SVC number (and, if appropriate, the extended SVC routing number). The system performs an SVCUPDTE function to update SVC processing so that it gives control to the newly added LPA routine when that particular SVC is issued.

If the request indicates an extended SVC but the actual SVC is not an extended SVC, an error return results.

If an extended SVC routing code exceeds the value that the system can handle, an error return results. For SVC 109, any value is valid. For SVCs 116, 122, and 137 the value is release-dependent.

The area specified by MODINFOX must contain the number of entries indicated by the NUMMOD parameter.

The default value is NO_MODINFOX.

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

,MASKDSNAME=*maskdsname*
,MASKDDNAME=*maskddname*
,MASKDCB=*maskdcb*
,MASKDCBPTR=*maskdcbptr*

When MODINFOTYPE=MEMBERMASK is specified, a required input parameter.

,MASKDSNAME=*maskdsname*

A parameter that contains the name of the data set/library from which all the input modules are to be loaded. The data set must be cataloged. It can be allocated as a PDS or PDSE program library.

Note that if the data set is migrated, the issuer's unit of work waits until the data set is retrieved before continuing.

If the caller is authorized only by the RACF FACILITY class resource (is not supervisor state, system key, system PKM, or APF-authorized), then the data set must be APF-authorized. Similarly, if the caller is supervisor state, system key, or system PKM, or APF-authorized, the data set need not be APF-authorized.

To code: Specify the RS-type address, or address in register (2)-(12), of a 44-character field.

,MASKDDNAME=*maskddname*

A parameter that contains the DDNAME of the data set/library (or concatenation of libraries) from which all the input modules are to be loaded. The system will open the DDNAME for input.

If the caller is authorized only by the RACF FACILITY class resource (is not supervisor state, system key, system PKM, or APF-authorized), then the concatenation must be APF-authorized.

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

,MASKDCB=*maskdcb*

A parameter that contains the opened DCB representing the data set/library (or concatenation of libraries) from which all the input modules are to be loaded. The DCB must be opened for input.

The caller must be either supervisor state, system key, system PKM, or APF-authorized to use this option.

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

,MASKDCBPTR=*maskdcbptr*

A parameter that contains the address of the opened DCB representing the data set/library (or concatenation of libraries) from which all the input modules are to be loaded. The DCB must be opened for input. You can specify MASKDCBPTR=CVTLINK to request that the LNKLST be used as the library concatenation (you must have an assembler USING established on the CVT data area in order to use this).

The caller must be either supervisor state, system key, system PKM, or APF-authorized to use this option, unless CVTLINK was specified.

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

,OUTAREAPTR=*outareaptr*

When MODINFOTYPE=MEMBERMASK is specified, a required output parameter that is to contain the address of an area obtained by the system. The area contains information about each member processed. It consists of contiguous entries, each mapped by DSECT LPMEA within macro CSVLPRET.

Note: If QUERYONLY=YES is specified, the output information never indicates "LpmeaSuccess" because the data returned for each matching member only provide information about requests that do not successfully complete.

It is expected that the caller will free this area, using either FREEMAIN or STORAGE RELEASE, after processing it. The area is in the key of the caller of CSVDYLPA. Its length is contained in the output field specified by the OutAreaLen parameter. Its subpool is contained in the output field specified by the OutAreaSP parameter.

This area is obtained on behalf of the caller, and so must be freed, whenever the return code is less than 8 (CsvdylpaRc_InvParm) and when the output field specified by the OutAreaNum parameter is non-zero.

Assuming that the caller is still running with the PSW key current when CSVDYLPA was issued, and specified OUTAREALEN=AREALEN, OUTAREASP=AREASP, and OUTAREAPTR=AREAPTR when issuing CSVDYLPA, the following can be used to free the area:

```
          ICM   1,15,AREAPTR
          BZ    NO_FREE
          FREEMAIN RU,A=(1),LV=AREALEN,SP=AREASP
NO_FREE   DS    0H
```

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

,OUTAREALEN=*outarealen*
When MODINFOTYPE=MEMBERMASK is specified, a required output parameter that is to contain the length of the output area.

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

,OUTAREANUM=*outareanum*
When MODINFOTYPE=MEMBERMASK is specified, a required output parameter that is to contain the number of entries in the output area.

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

,OUTAREASP=*outareasp*
When MODINFOTYPE=MEMBERMASK is specified, a required input/output parameter that on input contains the subpool to use for the output area. If the value is zero or specifies an unauthorized subpool, the system uses subpool 230. Since the storage obtained is expected to be in the key of the CSVDYLPA caller, the subpool must either be one that is key-specifiable or one that is only in the key of the CSVDYLPA caller. If it is not, the system uses subpool 230. On output, the field contains the subpool that was actually used.

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

,APFREQUIRED=YES | NO
An optional parameter that indicates whether or not the input data set or data set concatenation or (for the DCB representing the LNKLST, when system parameter LNKAUTH=APFTAB is in effect) the module's data set must be APF authorized. The data set / concatenation is specified by the input data set name, DCB, or DDNAME. This keyword is ignored unless the caller is supervisor state, system key, system PKM, or APF-authorized. In all other cases, APFREQUIRED=YES is used. If the module is to be fetched via pathname, the APFREQUIRED parameter does not apply; the module must be marked as an APF program via the UNIX extattr command with the +a attribute. The default is APFREQUIRED=YES.

,APFREQUIRED=YES
Indicates that the input data set or data set concatenation or (for the LNKLST when LNKAUTH=APFTAB| is in effect) the module's data set or (for the LNKLST when LNKAUTH=APFTAB is in effect) the module's data set must be APF-authorized.

,APFREQUIRED=NO
Indicates that the input data set or data set concatenation or (for the LNKLST when LNKAUTH=APFTAB is in effect) the module's data set need not be APF-authorized.

,SECMODCHECK=YES | NO
An optional parameter that indicates whether the RACF FACILITY class check should be done for the module or alias being added. This keyword is ignored unless the caller is supervisor state, system key, system PKM, or APF-authorized. In all other cases, SECMODCHECK=YES is used. The default is SECMODCHECK=YES.

,SECMODCHECK=YES
Indicates to do the RACF FACILITY class check. For a caller in supervisor state, system key, system PKM, or APF-authorized, the operation is allowed when the check results either in "success" or "no matching profile exists." For unauthorized callers, the operation is allowed only when the check results in "success".

,SECMODCHECK=NO
Indicates not to do the RACF FACILITY class check.

,REQUESTOR=*requestor*
A required input parameter that identifies the requester. This string is only used for helping to identify in a dump which requester caused the particular module to be added to LPA. Thus it is important that requesting products specify a value that does not match the value specified by another product. IBM requesters should begin the string with their component prefix.

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

,MODPROB=CONTINUE
,MODPROB=STOP
An optional parameter that indicates the action to be taken by the system if it encounters a problem processing an individual module entry (a condition that will not result in a return code greater than 4). This includes, but is not limited to, such cases as:

- The system could not locate the specified module or alias;
- The system could not allocate all the required storage; and
- The system could not load the module indicated by the directory entry.

The equate symbols with names beginning with "LpmeaModprob", in macro CSVLPRET, describe the full set of conditions.

When any problem, or a situation resulting in a return code of at least 8, is encountered, no modules are added to LPA. The default is MODPROB=CONTINUE.

,MODPROB=CONTINUE
Indicates to continue processing subsequent members in order to detect problems that are associated with those subsequent members. Any entry that is not successfully processed has the LpmeaModprob bit on, with additional information as indicated by the LpmeaModprobFunction field.

,MODPROB=STOP
Indicates to stop processing. The entry that is not successfully processed has the LpmeaModprob bit on, with additional information as indicated by the LpmeaModprobFunction field.

,ERRORDATA=*errordata*
An optional output parameter that contains additional diagnostic data for certain cases. In particular,

- For return code 8 (symbol CsvdylpaRc_InvParm) with reason code X'xxyy0829' (symbol CsvdylpaRsnBadAlloc), bytes 0-1 are the error reason code from the DYNALLOC service, bytes 2-3 are the information reason code from the DYNALLOC service, and bytes 4-7 are the SMS reason code from the DYNALLOC service (which is only relevant when the error reason code is of the form X'97xx').

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

,ADDALIAS=NO | YES
When BYADDR=NO and MODINFOTYPE=MEMBERLIST are specified, ADDALIAS is an optional parameter that indicates whether to add aliases of the input names.

The default is ADDALIAS=NO.

,ADDALIAS=NO
Indicates not to add aliases, but only process the input names provided.

,ADDALIAS=YES
Indicates to add aliases of the input names to the list to process.

Note: When adding is being done from a z/OS UNIX file, which is indicated by the PATHNAME keyword, ADDALIAS=YES is supported but cannot find aliases because this construct does not exist for z/OS UNIX files.

,OUTAREAPTR=*outareaptr*
When ADDALIAS=YES, BYADDR=NO and MODINFOTYPE=MEMBERLIST are specified, OUTAREAPTR is a required output parameter that is to contain the address of an area obtained by the system. This area contains information about each member that is processed. These members include the members and aliases of the members specified by the MODINFO, MODINFOX, and NUMMOD parameters. The area consists of contiguous entries each mapped by DSECT LPMEA in the CSVLPRET macro.

The caller must use the FREEMAIN or STORAGE RELEASE macro to free the area after processing it.

The area is in the key of the caller of CSVDYLPA. The length of the area is contained in the output field specified by the OUTAREALEN parameter. The subpool of the area is contained in the output field specified by the OUTAREASP parameter.

If the return code is 8 (CsvdylpaRc_InvParm) or less, and the output field specified by the OUTAREANUM parameter is not zero, the area is obtained on behalf of the caller and must be freed.

If the caller is running with the PSW key when CSVDYLPA was issued, and specified OUTAREALEN=AREALEN, OUTAREASP=AREASP, and OUTAREAPTR=AREAPTR, the following example shows how the area can be freed:

```
          ICM   1,15,AREAPTR
          BZ    NO_FREE
          FREEMAIN RU,A=(1),LV=AREALEN,SP=AREASP
                NO_FREE   DS    0H
```

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

,OUTAREALEN=*outarealen*
When ADDALIAS=YES, BYADDR=NO, and MODINFOTYPE=MEMBERLIST are specified, OUTAREALEN is a required output parameter that contains the length of the output area.

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

,OUTAREANUM=*outareanum*
When ADDALIAS=YES, BYADDR=NO, and MODINFOTYPE=MEMBERLIST are specified, OUTAREANUM is a required output parameter that contains the number of entries in the output area.

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

,OUTAREASP=*outareasp*
When ADDALIAS=YES, BYADDR=NO, and MODINFOTYPE=MEMBERLIST are specified, OUTAREASP is a required input/output parameter that on input contains the subpool to use for the output area.

If the value of OUTAREASP is zero or specifies an unauthorized subpool, the system uses subpool 230.

The storage obtained must be in the key of the CSVDYLPA caller. Therefore, the subpool must be key-specifiable or only in the key of the CSVDYLPA caller. If the subpool is not in the key of the CSVDYLPA caller, the system uses subpool 230.

On output, the OUTAREASP parameter contains the subpool that is used.

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

,ADDALIAS=NO | YES
When MODINFOTYPE=MEMBERMASK is specified, ADDALIAS is an optional parameter that indicates whether to add aliases of the input names.

The default is ADDALIAS=NO.

,ADDALIAS=NO
Indicates not to add aliases, but only process the input names provided.

,ADDALIAS=YES
Indicates to add aliases of the input names to the list to process.

,QUERYONLY=NO | YES
When MODINFOTYPE=MEMBERMASK is specified, QUERYONLY is an optional parameter that indicates whether to do the ADD operation or query the storage requirements of (E)CSA and (E)SQA.

The default is QUERYONLY=NO.

,QUERYONLY=NO
Indicates to do the ADD operation.

,QUERYONLY=YES
Indicates to do the QUERY operation only.

,AC1=YES | NO
When BYADDR=YES and MODINFOTYPE=MEMBERLIST are specified, AC1 is an optional parameter that indicates whether to treat the module as if it is bound with AC=1.

The default is AC1=YES.

,AC1=YES
Indicates to treat the module as AC=1.

,AC1=NO
Indicates to treat the module as AC=0.

,LONGPARMOK=NO | YES
When BYADDR=YES and MODINFOTYPE=MEMBERLIST are specified, LONGPARMOK is an optional parameter that indicates whether a parameter is allowed with more than 100 characters long, if this module is the jobstep program, or the target of execMVS, and if the module is to get control authorized.

The default is LONGPARMOK=NO.

,LONGPARMOK=NO
Indicates that long parm is not OK.

,LONGPARMOK=YES
Indicates that long parm is OK.

,LPMEAQ=*lpmeaq*
When QUERYONLY=YES and MODINFOTYPE=MEMBERMASK are specified, LPMEAQ is a required input parameter that specifies an area to contain the output information. The area is mapped by DSECT LPMEAQ in macro CSVLPRET.

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

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

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

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

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

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

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

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

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 319 for the return and reason codes.

Example

See “Example 1” on page 324 for an example.

REQUEST=DELETE option of CSVDYLPA

REQUEST=DELETE allows you to remove from LPA one or more modules or aliases that had previously been added by dynamic LPA services.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Any of the following: • Supervisor state • PKM 0-7 • PSW key 0-7 • APF authorized • In addition to any of the above, if SECMODCHECK=YES is specified or defaulted to, issuer must be authorized for UPDATE to the RACF FACILITY class resource CSVDYLPA.DELETE.modname
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided data set name through the DSNAME parameter has the same requirements and restrictions as the control parameters. The user-provided information through the MODINFO parameter has the same requirements and restrictions as the control parameters.

Programming requirements

The caller should include the CSVLPRET macro to get equate symbols for the return and reason codes.

The caller must include the CSVLPRET macro to get a mapping of the input/output area provided via the MODINFO

The caller may hold the system ENQ resource with QNAME SYSZCSV and RNAME CSVDYLPA in the exclusive state. While this ENQ resource is held, any other requests to use the CSVDYLPA services to ADD or DELETE will be delayed. The ENQ resource must not be held in the shared state when using CSVDYLPA services. parameter.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

Before issuing the CSVDYLPA macro, the caller does not need to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CSVDYLPA macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYLPA.
CSVDYLPA	
ƀ	One or more blanks must follow CSVDYLPA.
REQUEST=DELETE	
,MODINFO=*modinfo*	*modinfo:* RS-type address or address in register (2) - (12)
,NUMMOD=*nummod*	*nummod:* RS-type address or address in register (2) - (12)
,TYPE=BYTOKEN	**Default:** TYPE=BYTOKEN
,TYPE=CURRENT	
,TYPE=OLDEST	
,TYPE=VALUE	
,TYPEVALUE=*typevalue*	*typevalue:* RS-type address or register (2) - (12).
,SECMODCHECK=YES	**Default:** SECMODCHECK=YES
,SECMODCHECK=NO	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)

Syntax	Description
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

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

REQUEST=DELETE
A required parameter. REQUEST=DELETE indicates to remove one or more modules from LPA. You can only remove a module that has been added to LPA using dynamic LPA services. You cannot remove a module that was built into LPA during IPL.

REQUEST=DELETE must be used with extreme caution, as the system does not keep track of whether or not any code is currently running within the code that is to be deleted. It is up to the caller to request deletion at an appropriate point.

If the module was added with BYADDR=YES, the system will not free the storage for the module. Otherwise, the system will free the storage for the module when there are no longer any major names or aliases associated with that storage.

,MODINFO=*modinfo*
A required input/output parameter that specifies an area that contains contiguous entries. Each entry contains a module name, a delete token (when TYPE=BYTOKEN is specified or defaulted), and status flags. Each entry is mapped by DSECT LPMED in macro CSVLPRET. The number of entries must match the value provided via the nummod parameter. The flags area in each entry should be cleared before calling the CSVDYLPA service.

When deleting a module using TYPE=BYTOKEN, you use the token returned by CSVDYLPA REQUEST=ADD for that module (placing it in the LpmedDeleteToken field of the LPMED DSECT).

On output, among other possibilities, the status area might indicate:

- The module had not been added using dynamic LPA services;
- You are not authorized to process the particular module.

For a complete list of the possible problem types, refer to the equate symbols beginning with LpmedModprob in the CSVLPRET data area.

Within each LPMED, the module name should be left-justified padded on the right with EBCDIC blanks if less than 8 characters long. There should be no embedded blanks.

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

,NUMMOD=*nummod*
A required input parameter that contains the number of entries in the area specified by the modinfo parameter. Nummod must be in the range 1-256.

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

,TYPE=BYTOKEN
,TYPE=CURRENT
,TYPE=OLDEST
,TYPE=VALUE
An optional parameter that indicates the type of deletion requested. The default is TYPE=BYTOKEN.

,TYPE=BYTOKEN
indicates that the delete token is provided, and identifies the instance of the module to delete.

,TYPE=CURRENT
indicates that the most current instance of the module that was added by dynamic LPA services is to be deleted.

,TYPE=OLDEST
indicates that the oldest instance of the module that was added by dynamic LPA services, other than the current one, is to be deleted.

,TYPE=VALUE
indicates that the processing depends on the value provided by the TYPEVALUE parameter.

,TYPEVALUE=*typevalue*
When TYPE=VALUE, a required input parameter that contains the TYPE value. You can get equates to use when setting *typevalue* by using the list form of the CSVDYLPA macro. For example, an invocation of CSVDYLPA MF=(L,LNAME) produces equates LNAME_XTYPE_BYTOKEN, LNAME_XTYPE_CURRENT, and LNAME_XTYPE_OLDEST.

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

,SECMODCHECK=YES
,SECMODCHECK=NO
An optional parameter that indicates whether or not the RACF FACILITY class check should be done for the module or alias being deleted. This keyword is ignored unless the caller is supervisor state, system key, system PKM, or APF authorized. In all other cases, SECMODCHECK=YES is used. The default is SECMODCHECK=YES.

,SECMODCHECK=YES
indicates to do the RACF FACILITY class check. For a caller in supervisor state, system key, system PKM, or APF authorized, the operation is allowed when the check results either in "success" or "no matching profile exists." For unauthorized callers, the operation is allowed only when the check results in "success".

,SECMODCHECK=NO
indicates not to do the RACF FACILITY class check.

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

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

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

- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

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

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 319 for the return and reason codes.

Example

See “Example 1” on page 324 for an example.

REQUEST=DEFLPAWAIT option of CSVDYLPA

REQUEST=DEFLPAWAIT allows you to request that your work unit wait until processing of LPA statements in the IPL-time PROGxx parmlib members (deferred LPA) complete.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Any of the following: • Supervisor state • PKM 0-7 • PSW key 0-7 • APF authorized
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

Before issuing the CSVDYLPA macro, the caller does not need to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code if GPR15 is not 0

1
Used as a work register by the system

2–13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CSVDYLPA macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYLPA.
CSVDYLPA	
ƀ	One or more blanks must follow CSVDYLPA.
REQUEST=DEFLPAWAIT	
,RETCODE=*retcode*	*retcode:* RS-type address or address in register (2) - (12) or (15), (GPR15)
,RSNCODE=*rsncode*	*rsncode:* RS-type address or address in register (0) or (2) - (12), (00), (GPR)
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	

Syntax	Description
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

Parameters

The parameters are explained as follows:

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

REQUEST=DEFLPAWAIT
A required parameter. REQUEST=DEFLPAWAIT indicates to wait for processing of LPA statements in the IPL-time PROGxx parmlib members (deferred LPA) to complete. If deferred LPA is complete, the system returns control immediately. You can use the QUERYDEFLPA request to query whether this call is required.

,RETCODE=*retcode*
An optional output parameter that stores the return code copied from GPR 15. If you specify 15, GPR15, REG15, or R15 (within or without parentheses), the value remains 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 that stores the return code copied from GPR 0. If you specify 0, 00, GPR0, GPR00, REG0, REG00, or R0 (within or without parentheses), the value remains 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**, is the lowest version that allows all parameters specified on the request to be processed. If the PLISTVER parameter is omitted, IMPLIED_VERSION is the default.
- **MAX**, allows the parameter list to be the largest size possible. This size might increase in future releases and affect the amount of storage that your program needs.

  You can specify PLISTVER=MAX on the list form of the macro to ensure that the list form parameter list can hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. Specifying PLISTVER=MAX ensures that the parameter list does not overwrite nearby storage.
- **0**, if you use the available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr*,*attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)

An optional input parameter that specifies the macro form.

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

Use MF=L to specify the list form of the macro. Use the list form together with the execute form of the macro for applications that require reentrant code. The list form defines an area of storage that the execute form uses to store the parameters. Only the PLISTVER parameter can 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 - 60 character input string that you can use to force boundary alignment of the parameter list. Use a value of 0F to force the parameter list to a word boundary, or 0D to force the parameter list to a doubleword boundary. If you do not code *attr*, the system provides a value of 0D.

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

ABEND codes

None.

Return codes

See "Return and reason codes" on page 319 for the return and reason codes.

Examples

See "Example 2" on page 325 for an example.

REQUEST=QUERYDYN option of CSVDYLPA

REQUEST=QUERYDYN allows you to query whether the ADD and DELETE functions are available.

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:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
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

The caller must include the CSVLPRET macro to get equates for the information returned via the DYNFUNC parameter.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

Before issuing the CSVDYLPA macro, the caller does not need to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
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

None.

Syntax

The CSVDYLPA macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYLPA.
CSVDYLPA	
ƀ	One or more blanks must follow CSVDYLPA.
REQUEST=QUERYDYN	
,DYNFUNC=*dynfunc*	*dynfunc:* RS-type address or address in register (2) - (12)

Parameters

The parameters are explained as follows:

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

REQUEST=QUERYDYN
A required parameter. REQUEST=QUERYDYN indicates to return an indication of whether modules can be added to or deleted from LPA.

,DYNFUNC=*dynfunc*
A required output parameter that will contain the availability of the ADD and DELETE functions. If 0 (symbol CsvdylpaDynNotAvailable in macro CSVLPRET) the functions are not available. If 1 (symbol CsvdylpaDynAvailable) the functions are available.

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

ABEND codes

None.

Return codes

None.

Examples

None.

REQUEST=QUERYDEFLPA option of CSVDYLPA

REQUEST=QUERYDEFLPA allows you to query whether processing of LPA statements in the IPL-time PROGxx parmlib members (deferred LPA) is complete.

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:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

The caller must include the CSVLPRET macro to get equates for the information returned by the DEFLPASTATE parameter.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

Before issuing the CSVDYLPA macro, the caller does not need to place any information into any general purpose register (GPR) or access register (AR) unless using it in register notation for a particular parameter, or using it as a base register.

Output register information

When control returns to the caller, the GPRs contain:

Register
Contents

0–1
Used as work registers by the system

2–13
Unchanged

14–15
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

None.

Syntax

The CSVDYLPA macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYLPA.
CSVDYLPA	
ƀ	One or more blanks must follow CSVDYLPA.
REQUEST=QUERYDEFLPA	
,DEFLPASTATE=*deflpastate*	*deflpastate:* RS-type address or address in register (2) - (12)

Parameters

The parameters are explained as follows:

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

REQUEST=QUERYDEFLPA
A required parameter. REQUEST=QUERYDEFLPA indicates whether processing of LPA statements in the IPL-time PROGxx parmlib members (deferred LPA) is complete.

,DEFLPASTATE=*deflpastate*
A required output parameter that is to contain the deferred LPA completion state.

If the value of DEFLPASTATE is 0 (symbol CsvdylpaDefLPA_INCOMPLETE in macro CSVLPRET), processing is not complete. You can issue CSVDYLPA REQUEST=DEFLPAWAIT to wait for completion. If you are not authorized to issue CSVDYLPA REQUEST=DEFLPAWAIT, you can use the CSVDLPAW routine.

If the value of DEFLPASTATE is 1 (symbol CsvdylpaDefLPA_COMPLETE in macro CSVLPRET), processing is complete. In this case, you do not have to issue CSVDYLPA REQUEST=DEFLPAWAIT or use the CSVDLPAW routine.

From a program, you can issue LINK EP=CSVDLPAW to wait for completion. You can also have a JCL step with EXEC PGM=CSVDLPAW prior to the step with your program, and then when your program gets control, deferred LPA processing will be complete.

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

ABEND codes

None.

Return codes

None.

Examples

See “Example 2” on page 325 for an example.

Chapter 38. CSVDYNEX — Provide dynamic exits services

Description

The CSVDYNEX macro defines exits. It also controls their use and associates exit routines with those exits. You might be familiar with system installation exits that offer an installation an opportunity to modify the system's own processing. CSVDYNEX allows you to offer exits within your own programs. Additionally, the CSVDYNEX macro allows you to associate exit routines with system exits, such as the SMF and allocation exits.

As used here, an exit point is a location in a program's processing where the system transfers control to (or calls) another piece of code, known as an exit routine. An exit routine can give information to the caller that allows the caller to do additional processing. An exit is simply a set of information that includes:

- Criteria for exit routines that are to get control at the exit point
- Directions for how the system is to transfer control to an exit routine, process the exit routine, and handle recovery.

There are ten CSVDYNEX requests, issued through the REQUEST parameter on CSVDYNEX; for example, you issue the LIST request by specifying CSVDYNEX REQUEST=LIST with appropriate parameters.

Through the DEFINE request, you define an exit; that is, you give the exit a unique name and specify its characteristics. Through the ADD request, you add or associate an exit routine with an exit. More than one exit routine can be associated with an exit. The location of the exit point (the point at which control passes to the exit routine) is determined by the placement of the CALL request. The CALL request names the exit; the system finds the set of information known as the exit, and finds the exit routine or routines that are associated with the exit. The system then passes control to those exit routines; it processes them, handles any recovery, and returns control to the caller. It performs these actions according to information you provide on the DEFINE, ADD, and CALL requests.

The MODIFY and ATTRIB requests make certain changes to the exit routines and the exits.

The DELETE request deletes or disassociates an exit routine from an exit. The UNDEFINE request **removes** the exit from the system.

The LIST and QUERY requests return information about exits and exit routines.

The RECOVER request provides recovery for an exit routine that is called with FASTPATH processing in effect.

For ease of use, the standard form of the macro is shown for each CSVDYNEX request. The ten requests are described on the following pages, with the standard form syntax diagrams, descriptions of the parameters, environment, programming requirements, and restrictions:

- Defining an exit, in “Define an exit” on page 359
- Adding an exit routine to an exit, in “Add an exit routine to an exit” on page 366
- Changing the state of an exit routine, in “Change the state of an exit routine” on page 373
- Deleting an exit routine from an exit, in “Delete an exit routine from an exit” on page 376
- Removing the definition of an exit, in “Remove the definition of an exit” on page 379
- Changing the attributes of an exit, in “Change the attributes of an exit” on page 382
- Listing information about one or more exits, in “List information about one or more exits” on page 385
- Calling one or more exit routines at an exit, in “Call one or more exit routines at an exit” on page 389
- Providing recovery for an exit routine that has abnormally ended, in “Provide recovery for an exit routine that abnormally ended” on page 395

- Determining whether an exit routine exists for an exit, in "Determine whether an exit routine exists for an exit" on page 399.

Following the descriptions of the standard forms of all requests are:

- The return and reason codes, in "Return and reason codes" on page 407
- Examples of using CSVDYNEX, in "Examples of the CSVDYNEX macro" on page 414
- The list form, in "CSVDYNEX - List form" on page 423
- The modify form, in "CSVDYNEX - Modify form" on page 424
- The execute form, in "CSVDYNEX - Execute form" on page 425.

Input register information for CSVDYNEX

With the exception of the CSVDYNEX QUERY request and the CSVDYNEX CALL request with FASTPATH=YES, 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.

Before issuing the CSVDYNEX QUERY request or the CSVDYNEX CALL request with FASTPATH=YES, the caller must ensure that the following GPR contains the specified information:

Register
Contents

13
Address of a standard 72-byte save area in the primary address space. You can use this same address for GPR 13 in the register update block (RUB). For more information about the RUB, see the description of the RUB parameter on the CSVDYNEX CALL request.

Output register information for CSVDYNEX

When control returns to the caller, the GPRs contain:

Register
Contents

0
Reason code, if GPR 15 is not zero

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

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

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 using a service. If the system changes the contents of registers on which the caller depends, the caller must save them before calling the service, and restore them after the system returns control.

Performance implications

None.

Define an exit

The CSVDYNEX DEFINE request provides the set of information known as the exit. To define an exit, you:

- Give the exit a name (EXITNAME parameter)
- Establish the persistence of the exit (PERSIST parameter)
- Set defaults for how the system is to handle the abnormal ending of an exit routine (ABENDNUM and ABENDCONSEC parameters)
- Establish how the system processes exit routines (FASTPATH, KEY, and ANYKEY parameters)
- Specify how the system is to handle the return codes from exit routines (RCFROM, RCCOMPARE, RCTO, RCCVAL, and CALLSTOPRC parameters)
- Establish some requirements for the exit routine or routines that the system invokes at the exit:
  - Addressing mode (AMODE parameter)
  - Reentrancy (REENTRANT parameter)

The CSVDYNEX UNDEFINE request removes the definition that is established by the DEFINE request.

Note: You define an exit implicitly when you:

- Add exit routines to an exit before the exit has been defined
- Set attributes for an exit using the ATTRIB parameter before the exit has been defined.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF UPDATE authority to FACILITY class entity CSVDYNEX.exitname.DEFINE.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.

Environmental factor	Requirement
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See "Input register information for CSVDYNEX" on page 358 for input register information for the CSVDYNEX DEFINE request.

Output register information

See "Output register information for CSVDYNEX" on page 358 for output register information for the CSVDYNEX DEFINE request.

Syntax

The standard form of the DEFINE request on the CSVDYNEX macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
REQUEST=DEFINE	
,EXITNAME=*exitname*	*exitname*: RS-type address or address in register (2) - (12).
,AMODE=31	**Default:** AMODE=31

Syntax	Description
,AMODE=24	
,AMODE=DEFINED	
,REENTRANT=OPT	**Default:** REENTRANT=OPT
,REENTRANT=REQ	
,PERSIST=TASK	**Default:** PERSIST=TASK
,PERSIST=ADDRESSSPACE	
,PERSIST=IPL	
,ABENDNUM=*abendnum*	*abendnum*: RS-type address or address in register (2) - (12).
,ABENDCONSEC=NO	**Default:** ABENDCONSEC=NO
,ABENDCONSEC=YES	
,DISABLEDCALL=NOT_OK	**Default:** DISABLEDCALL=NOT_OK
,DISABLEDCALL=OK	
,FASTPATH=NO,KEY=ZERO	
	Default: FASTPATH=NO,KEY=ZERO
,FASTPATH=NO,KEY=*key*	*key*: RS-type address or address in register (2) - (12).
,FASTPATH=NO	
,FASTPATH=YES,KEY=*key*	
,FASTPATH=YES,ANYKEY=NO,KEY=*key*	
,FASTPATH=YES,ANYKEY=YES	
,LOADAPF=NO	**Default:** LOADAPF=NO
,LOADAPF=Yes	
,RCFROM=*rcfrom*,RCCOMPARE=*option*,	
RCTO=*rcto*	
,RCFROM=*rcfrom*,RCCOMPARE=VALUE,	
RCCVAL=*rccval*,RCTO=*rcto*	

Syntax	Description
	rcfrom: RS-type address or address in register (2) - (12).
	option: See the RCCOMPARE parameter description.
	rcto: RS-type address or address in register (2) - (12).
	rccval: RS-type address or address in register (2) - (12).
,CALLSTOPRC=*callstoprc*	*callstoprc*: RS-type address or address in register (2) - (12).
EXITTYPE=UNSPECIFIED	**Default:** EXITTYPE=UNSPECIFIED
EXITTYPE=INSTALLATION	
EXITTYPE=PROGRAM	
,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=DEFINE
Defines an exit.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit. Names of fewer than 16 characters must be left-justified in the 16-character field and padded with blanks.

Names must be unique within the system. To avoid the names the system uses, begin the name with the letters J through Z (but never with the character string “SYS”). Other rules are:

- You can use alphanumerics, underscores, and periods.
- Do not use imbedded blanks.
- Do not begin with X'00' or a blank.

IBM recommends that you specify exit names using upper case; the EXIT statement of the PROGxx parmlib member and the commands (SETPROG, SET PROG=, and DISPLAY PROG) used to control dynamic exits require upper case.

,AMODE=31
,AMODE=24
,AMODE=DEFINED
Specifies the addressing mode of exit routines that the exit point is to invoke. AMODE=31, the default, specifies that the exit routines must have an AMODE of 31-bit. AMODE=24 specifies that the AMODE must be 24-bit. AMODE=DEFINED indicates that the exit routine takes control in the AMODE that was defined for the exit routine's load module at the time of the linkedit (for example, by a LOAD action).

Note that if you use AMODE=24 or AMODE=DEFINED, you cannot specify GPR 2 in the register update block (RUB) on the CSVDYNEX CALL request.

,REENTRANT=OPT
,REENTRANT=REQ
Specifies whether exit routines to be added to the exit can optionally be reentrant or are required to be reentrant. The default is REENTRANT=OPT. If you specify REENTRANT=REQ, the CSVDYNEX service verifies that exit routines added with the CSVDYNEX ADD request have been linkedited with the RENT attribute.

If you specify REENTRANT=OPT, you cannot specify ANYKEY=YES.

,PERSIST=TASK
,PERSIST=ADDRESSSPACE
,PERSIST=IPL
Specifies the persistence of the exit in relationship to the issuer of the CSVDYNEX DEFINE request that defines the exit. PERSIST indicates the following:

- PERSIST=TASK, the default, indicates that the exit exists only as long as the task of the issuer.
- PERSIST=ADDRESSSPACE indicates that the exit exists only as long as the address space of the issuer.
- PERSIST=IPL indicates that the exit exists for the duration of the IPL.

,ABENDNUM=*abendnum*
ABENDNUM specifies a fullword area (or a register containing the address of a fullword area) that contains the number of abnormal endings an exit routine can have before it becomes inactive. An inactive exit routine is one that is associated with an exit, but will not be called. For example, if you specify the value *n*, the exit routine becomes inactive when the *nth* abnormal ending occurs. The value you specify for ABENDNUM is interpreted as a signed, 31-bit number.

If you omit ABENDNUM or specify a value of 0 or 1, the exit routine becomes inactive the first time it abnormally ends. Use the ABENDCONSEC parameter to establish whether *n* means consecutive abnormal endings or cumulative abnormal endings.

The ADDABENDNUM parameter on the CSVDYNEX ADD request overrides ABENDNUM.

Use the CSVDYNEX MODIFY request to change the state of the exit routine from inactive to active.

,ABENDCONSEC=NO
,ABENDCONSEC=YES
Specifies whether the number of total abnormal endings you specify on ABENDNUM is to be cumulative or consecutive.

- ABENDCONSEC=NO means that after the specified number of accumulated abnormal endings, the exit routine becomes inactive.
- ABENDCONSEC=YES means that after the specified number of consecutive abnormal endings, the exit routine becomes inactive. An exit that is defined as FASTPATH=YES and ABENDCONSEC=YES cannot be defined with either PSW key 8 to 15, or with ANYKEY=YES.

,DISABLEDCALL=NOT_OK
,DISABLEDCALL=OK
When FASTPATH=YES and REQUEST=DEFINE are specified, an optional parameter that indicates whether the exit routines can be called while disabled for I/O and external interrupts. The default is DISABLEDCALL=NOT_OK.

,DISABLEDCALL=NOT_OK
indicates that the exit routines can not be called while disabled.

,DISABLEDCALL=OK
indicates that the exit routines can be called while disabled. Exit routines for this exit must be in page-fixed storage. If the dynamic exits facility is responsible for loading the exit routine, it will use page-fixed storage. Otherwise, it is the exit adder's responsibility to make sure that this is done. Failure to do so may result in abends or unpredictable results.

,FASTPATH=NO,KEY=ZERO
,FASTPATH=NO,KEY=*key*
,FASTPATH=NO
,FASTPATH=YES,KEY=*key*
,FASTPATH=YES,ANYKEY=NO,KEY=*key*
,FASTPATH=YES,ANYKEY=YES
Specifies how the system is to process the CALL request. On the CSVDYNEX DEFINE request, the FASTPATH keyword enables the FASTPATH function. On the CSVDYNEX CALL request, it specifies whether or not to use the function. With FASTPATH=YES, processing is faster because:

- The system does not provide normal recovery for the exit routine and does no SAF authorization checking.
- The exit routine runs in the PSW key and with the authorization (problem state or supervisor state) of the caller.
- In its processing of the CALL request, the system uses a work area obtained by the issuer of the CALL or RECOVER request, instead of obtaining and releasing one. (See the WORKAREA parameter on the CSVDYNEX CALL and RECOVER requests.)

FASTPATH=NO, the default, specifies that FASTPATH processing does not apply to the exit. If you specify FASTPATH=NO on the DEFINE request, all subsequent CALL requests must also specify FASTPATH=NO.

The KEY=ZERO or KEY=*key* parameter specifies the storage key in which the system is to place the exit routine. KEY=*key* specifies a fullword (or a register containing the address of a fullword). The KEY parameter with FASTPATH=NO applies to non-reentrant exit routines only. The system places reentrant routines in storage key zero.

The KEY parameter is required with FASTPATH=YES, unless ANYKEY=YES is specified. The value must be 0 through 15. On a CSVDYNEX CALL request that specifies FASTPATH=YES and the KEY parameter, the PSW key of the caller must be the same as the value specified on the KEY parameter or must be 0.

ANYKEY=YES specifies that the exit routine may be called in any PSW key. The following restrictions apply when using ANYKEY=YES:

- You must specify REENTRANT=REQ
- You cannot specify ABENDCONSEC=YES
- You must specify FORCE=YES to delete an exit routine associated with an exit that has been defined with ANYKEY=YES.

LOADAPF=NO
LOADAPF=YES
This optional keyword specifies if every routine for the exit needs to come from an APF-authorized library when the routine is not found in LPA or IEANUC01.

Note: This keyword is only for users who issue CSVDYNEX REQUEST=DEFINE or CSVDYNEX REQUEST=ADD with either supervisor state and system key, or with APF-authorization. For other callers, the routine must come from an APF-authorized library.

- LOADAPF=NO means that the module can come from a non-APF-authorized library if the module is not found in LPA or IEANUC01.
- LOADAPF=YES means that the module must come from an APF-authorized library if the module is not found in LPA or IEANUC01.

The default is LOADAPF=NO.

,RCFROM=*rcfrom*,RCCOMPARE=*option*,RCTO=*rcto*
,RCFROM=*rcfrom*,RCCOMPARE=VALUE,RCCVAL=*rccval*,RCTO=*rcto*

Specifies how the system is to process return codes from the exit routines that are associated with the exit.

RCFROM=*rcfrom* specifies a fullword (or a register containing the address of a fullword) that contains the value known as the RCFROM return code. The system compares the return code from each module called for this exit to the RCFROM return code, using the comparison designated by RCCOMPARE.

RCCOMPARE=*option* indicates the type of comparison the system is to make. The options on RCCOMPARE are:

- EQ - equal
- NE - not equal
- GT - greater than
- LT - less than
- GE - greater than or equal to
- LE - less than or equal to
- VALUE - the type of comparison is specified on the RCCVAL parameter.

RCTO=*rcto* specifies a fullword (or a register containing the address of a fullword) that contains a value that the system will substitute for the actual return code if the comparison is favorable.

RCCVAL specifies a one-byte field (or a register containing the address of a one-byte field) that contains the value that indicates the type of comparison the system is to do when comparing the actual return code with the value provided in RCFROM. The constants produced by the list form of the macro can be used. For example, CSVDYNEX MF=(L,MYLIST) would produce such equate symbols as MYLIST_XRCCOMPARE_EQ and MYLIST_XRCCOMPARE_GT.

RCCVAL applies only to RCCOMPARE=VALUE, and is required with that parameter.

The default for the RCFROM parameter and related parameters is that the system does no matching or substituting of the return code.

,CALLSTOPRC=*callstoprc*

Specifies a fullword (or a register containing the address of a fullword) that contains a value that the system is to compare with an exit routine's return code. If the exit routine's return code matches the value you specify on CALLSTOPRC, no more exit routines will be called at that exit for the life of this CALL request.

The default for CALLSTOPRC is that the system does no return code comparison.

,EXITTYPE=UNSPECIFIED
,EXITTYPE=INSTALLATION
,EXITTYPE=PROGRAM

When REQUEST=DEFINE is specified, EXITTYPE is an optional parameter that indicates the type of exit.

No enforcement is made with respect to the specified exit type, but the DISPLAY PROG,EXIT command allows you to request to limit the display to exits of a particular type.

The default is EXITTYPE=UNSPECIFIED

EXITTYPE=UNSPECIFIED indicates that the exit type is unspecified.

EXITTYPE=INSTALLATION indicates that this is an installation exit to be managed in part by the SETPROG or PROGxx parmlib member.

EXITTYPE=PROGRAM indicates that this is a program exit to be managed by the CSVDYNEX macro.

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See "Return and reason codes" on page 407 for the return and reason codes for the CSVDYNEX DEFINE request.

Example

For an example of how to define an exit, see "Example 1" on page 414.

Add an exit routine to an exit

Adding an exit routine to an exit means associating an exit routine with an exit that has already been defined or, in some cases, has not yet been defined. On the CSVDYNEX ADD request, you:

- Name the exit that the exit routine is to be associated with (EXITNAME parameter)
- Tell the system where to find the exit routine (MODNAME, DSNAME, and MODADDR parameters). Note that an exit routine must not be RMODE 64.
- Request that the system send a message to the operator if the system encounters certain error conditions when processing the exit routine (MESSAGE parameter)
- Define the initial state of the exit routine as active or inactive (STATE parameter)
- Specify a condition under which the exit routine is to get control: a particular job must be running (JOBNAME parameter), or a particular address space must be the primary address space (STOKEN parameter)
- Specify how many times an exit routine can abnormally end before it becomes inactive (ADDABENDNUM and ABENDCONSEC parameters)
- Control the order in which the new exit routine is called (POS parameter)
- Associate an exit routine with an existing system exit that is defined to the dynamic exits facility (see *z/OS MVS Installation Exits* for a list of the dynamic exits).

The CSVDYNEX DELETE request deletes an exit routine that was added to an exit by the CSVDYNEX ADD request.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF UPDATE authority to FACILITY class entity CSVDYNEX.exitname.modname.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters and the area that contains the data set name (specified on DSNAME) must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX ADD request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX ADD request.

Syntax

The standard form of the ADD request on the CSVDYNEX macro is written as follows:

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

Syntax	Description
␣	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
REQUEST=ADD	
,EXITNAME=*exitname*	*exitname*: RS-type address or register (2) - (12).
,MODNAME=*modname*	*modname*: RS-type address or register (2) - (12).
,STATE=ACTIVE	**Default:** STATE=ACTIVE
,STATE=INACTIVE	
,MESSAGE=NO	**Default:** MESSAGE=NO
,MESSAGE=ERROR	
,MESSAGE=FOUNDBUTERROR	
,DSNAME=*dsname*	*dsname*: RS-type address or register (2) - (12).
,MODADDR=*modaddr*	*modaddr*: RS-type address or register (2) - (12).
,JOBNAME=ANY	**Default:** JOBNAME=ANY
,JOBNAME=*jobname*	*jobname*: RS-type address or register (2) - (12).
,STOKEN=*stoken*	*stoken*: RS-type address or register (2) - (12).
,ADDABENDNUM=UNCHANGED	
,ADDABENDNUM=*addabendnum*	
	Default: ADDABENDNUM=UNCHANGED

Syntax	Description
	addabendnum: RS-type address or register (2) - (12).
,ABENDCONSEC=NO	**Default:** ABENDCONSEC=NO
,ABENDCONSEC=YES	
,DELETEFORCE=NO	**Default:** DELETEFORCE=NO
,DELETEFORCE=YES	
,POS=SYSTEM	**Default:** POS=SYSTEM
,POS=FIRST	
,POS=LAST	
PARAM=*param*	*param*: RS-type address or register (2) - (12).
PARAM=*NO_PARAM*	**Default:** PARAM=*NO_PARAM*
,SERVICEMASK=*servicemask*	*servicemask*: RS-type address or address in register (2)-(12)
,SERVICEMASK=ALL_SERVICES	**Default:** SERVICEMASK=ALL_SERVICES
,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=ADD
Adds an exit routine to an exit.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that has been defined to the dynamic exits facility. If the name has fewer than 16 characters, left-justify the name and pad the field with blanks.

,MODNAME=*modname*
Specifies an 8-byte field (or a register containing the address of an 8-byte field) containing the name of an exit routine to be added to the exit. The first character must not be X'00' or blank. (If you specify MODADDR, *modname* is the name of the exit routine. If you do not specify MODADDR, *modname* designates a load module or alias, whose entry point is the starting address of the exit routine.) Names of fewer than 8 characters must be left-justified in the 8-byte field and padded with blanks. Note that an exit routine must not be RMODE 64.

,STATE=ACTIVE
,STATE=INACTIVE
Specifies the state you want the exit routine to have. An active exit routine is associated with an exit and will be called if the exit is invoked. An inactive exit routine is associated with an exit, but will not be called. To change the state of an exit routine, use the CSVDYNEX MODIFY request. The default is STATE=ACTIVE.

,MESSAGE=NO
,MESSAGE=ERROR
,MESSAGE=FOUNDBUTERROR
Specifies whether the system is to send message CSV431I to the operator if the system encounters certain errors when processing the exit routine.

- MESSAGE=NO, the default, requests that the system send the return and reason codes only.
- MESSAGE=ERROR requests that the system send the message if it encounters any of the following:
  - The exit requires reentrancy; the exit routine is not reentrant.
  - The exit requires AMODE=31, but the exit routine is AMODE=24; or the exit requires AMODE=24, but the exit routine is AMODE=31.
  - The exit routine could not be located.
  - A CSVDYNEX ADD request specified ABENDCONSEC=YES, but the exit is defined with FASTPATH=YES and KEY=key, where KEY is in the range 8 to 15, or with FASTPATH=YES and ANYKEY=YES.
  - A CSVDYNEX ADD request specified that the exit routine should be loaded from a particular data set, but that data set is not APF-authorized, and the caller is in problem state, with PSW key 8-15, and is not APF-authorized.
- MESSAGE=FOUNDBUTERROR requests that the system send message CSV431I if it encounters any of the circumstances covered by MESSAGE=ERROR, except the following circumstance:
  - The exit routine could not be located.

,DSNAME=*dsname*
,MODADDR=*modaddr*
Tells the system how to locate the exit routine to be added. If you specify neither DSNAME nor MODADDR, the system will try to locate the module using LPA, the LNKLST concatenation, and the nucleus.

DSNAME specifies a field (or a register containing the address of a field) containing the 44-character name of a data set or library from which the module is to be obtained. Some rules for specifying DSNAME are:

- You can allocate the data set as a PDS or a PDSE.
- If the library name contains fewer than 44 characters, left-justify the name in a 44-character field and pad it with blanks.
- If you specify a data set name that begins with a blank or X'00', the system responds as if you had specified no data set.
- Specify DSNAME only if dynamic allocations are enabled within the caller's primary address space.
- If the caller is in problem state with PSW key 8 to 15, and is not APF-authorized, the data set must be APF-authorized. Otherwise, the data set does not need to be APF-authorized.
- The data set must be cataloged.

If the data set has been migrated, your program will have to wait for the system to retrieve it.

MODADDR specifies a fullword (or a register containing the address of a fullword) that contains the address of the exit routine to be added. If the exit routine is to get control in 31-bit mode, bit 0 should be on; if in 24-bit mode, bit 0 should be off. The system assumes that the designated exit routine is reentrant.

MODADDR cannot be used if the caller is in problem state with PSW key 8 to 15 and is not APF-authorized.

If you specify MODADDR, make sure the subpool and the key in which the exit routine resides are appropriate for the address spaces and keys in which the exit routine can get control. For example, if you specify STOKEN, the exit routine can reside in the private area of the address space designated by STOKEN. If you do not specify STOKEN, and the exit routine can be called from other address spaces, make sure the exit routine resides in the common area.

Note that an exit routine must not be RMODE 64.

If the storage is fetch-protected, the storage key must not conflict with the PSW key on entry to the exit routine. To prevent accidental modification by unauthorized users, the storage for exit routines that get control in system key must not be PSW key 8-15.

,JOBNAME=ANY
,JOBNAME=*jobname*
,STOKEN=*stoken*

Specifies a condition under which the exit routine is to get control. You can require that the exit routine take control:

- During the execution of a specific job, or jobs
- While a specific address space is the primary address space.

JOBNAME specifies an area (or a register containing the address of an area) that contains the 8-character name of the job that must be running at the time the exit routine is to get control. If the name has fewer than 8 characters, left-justify the name and pad the field with blanks. To indicate the name of more than one job, use an asterisk for the last non-blank character. A matching jobname is one that matches all the characters preceding the asterisk. The default is JOBNAME=ANY.

To indicate that the exit routine is not to be restricted to a particular job, specify a jobname of C'* '. To leave the jobname unchanged, specify a jobname with the first character X'00' or blank.

STOKEN specifies an area (or a register containing the address of an area) that contains the 8-character STOKEN of the address space that must be the primary address space at the time the exit routine receives control.

If you specify a jobname of C'* ', a jobname with the first character X'00', or JOBNAME=ANY, you are requesting that the system not check for the jobname or the STOKEN value.

,ADDABENDNUM=UNCHANGED
,ADDABENDNUM=*addabendnum*

Specifies how many times an exit routine can abnormally end before it becomes inactive. ADDABENDNUM=UNCHANGED, the default, specifies that the system will use the value specified on the ABENDNUM parameter on the CSVDYNEX DEFINE request that defines the exit, or the default value.

ADDABENDNUM=*addabendnum* specifies an area (or a register containing the address of an area) that contains the number of times an exit routine can end abnormally before it becomes inactive. For example, if you specify the value *n*, the exit routine becomes inactive when the *nth* abnormal ending occurs.

If you omit ADDABENDNUM or specify a value of 0, the exit routine becomes inactive on the basis of the ABENDNUM value specified on the CSVDYNEX DEFINE request. Use the ABENDCONSEC parameter to establish whether *n* means consecutive abnormal endings or cumulative abnormal endings.

The value you specify for ADDABENDNUM is interpreted as a signed, 31-bit number.

,ABENDCONSEC=NO
,ABENDCONSEC=YES

Specifies whether the number of abnormal endings you specify on ADDABENDNUM is to be cumulative or consecutive. The ABENDCONSEC parameter can be specified only if ADDABENDNUM is specified on this request. It will be ignored if the value you specify for ADDABENDNUM is 0.

- ABENDCONSEC=NO means that after the specified number of accumulated abnormal endings of the exit routine, the exit routine becomes inactive.
- ABENDCONSEC=YES means that after the specified number of consecutive abnormal endings of the exit routine, the exit routine becomes inactive. You cannot specify ABENDCONSEC=YES when adding an exit routine to an exit that is defined with PSW key 8 to 15 or ANYKEY=YES.

Note that the default is ABENDCONSEC=NO, which will override what was specified for ABENDCONSEC on the DEFINE request for this exit.

You can only specify the ABENDCONSEC parameter if you specify the ADDABENDNUM parameter.

,DELETEFORCE=NO
,DELETEFORCE=YES

Specifies whether any deletion of the exit routine (by CSVDYNEX, SETPROG, or an EXIT DELETE statement in PROGxx) must specify FORCE=YES:

- DELETEFORCE=NO, the default, specifies that exit routine deletion does **not** require FORCE=YES.
- DELETEFORCE=YES specifies that exit routine deletion does require FORCE=YES. If the deletion request does not specify FORCE=YES, the request is rejected.

,POS=SYSTEM
,POS=FIRST
,POS=LAST

Specifies the order in which the system calls the exit routine.

- POS=SYSTEM, the default, specifies that the exit routine may be called in any order relative to other routines associated with this exit.
- POS=FIRST specifies that the system should call the exit routine before any other routines associated with this exit, unless another exit routine, added after it, also specifies FIRST.
- POS=LAST specifies that the system should call the exit routine after any routines associated with this exit, unless other exit routines are added after it.

,PARAM=*param*
,PARAM=*NO_PARAM*

When REQUEST=ADD is specified, PARAM is an optional input parameter that specifies parameter information to be passed to the exit routine.

The first four bytes of the area are passed to the exit routine in access register 0. The second four bytes are passed to the exit routine in access register 1. If PARAM is not provided, zeroes are placed into the access registers.

The value is displayed using DISPLAY PROG,EXIT with the DIAG option. The display treats the 8 byte value as any other printable string so it is suggested that the 8 bytes be printable characters. If they are not, the only ramification is that they might not be visible via that display command.

The default is NO_PARAM.

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

,SERVICEMASK=*servicemask*
,SERVICEMASK=ALL_SERVICES

An optional input parameter containing the mask for this exit routine, to be applied to the service ID of the exit caller. Only if the ANDed value of the service mask and the service ID is nonzero for a given call and a given exit routine will that exit routine be called. The default is ALL_SERVICES.

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

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See "Return and reason codes" on page 407 for the return and reason codes for the CSVDYNEX ADD request.

Example

For an example of how to associate an exit routine with an exit, see "Example 2" on page 415.

Change the state of an exit routine

The state of an exit routine is active or inactive. An active exit routine is associated with an exit and will be called if the exit is called. An inactive exit routine is associated with an exit, but will not be called if the exit is called. The CSVDYNEX MODIFY request changes the state of an exit routine and, additionally, asks the system to check one of two conditions before it calls an exit routine. On the CSVDYNEX MODIFY request, you:

- Identify the exit (EXITNAME parameter) and the exit routine (MODNAME parameter)
- Specify the state you want to change to (STATE parameter)
- Specify the condition under which the exit routine is to get control (JOBNAME or STOKEN parameters).

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF UPDATE authority to FACILITY class entity CSVDYNEX.exitname.modname.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts

Environmental factor	Requirement
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 space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See "Input register information for CSVDYNEX" on page 358 for input register information for the CSVDYNEX MODIFY request.

Output register information

See "Output register information for CSVDYNEX" on page 358 for output register information for the CSVDYNEX MODIFY request.

Syntax

The standard form of the MODIFY request on the CSVDYNEX macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
REQUEST=MODIFY	
,EXITNAME=*exitname*	*exitname*: RS-type address or address in register (2) - (12).

Syntax	Description
,MODNAME=*modname*	*modname*: RS-type address or address in register (2) - (12).
,STATE=UNCHANGED	
,STATE=ACTIVE	
,STATE=INACTIVE	
,JOBNAME=*jobname*	*jobname*: RS-type address or address in register (2) - (12).
,STOKEN=*stoken*	*stoken*: RS-type address or address in register (2) - (12).
,SERVICEMASK=*servicemask*	*servicemask*: RS-type address or address in register (2)-(12)
,SERVICEMASK=ALL_SERVICES	**Default:** SERVICEMASK=ALL_SERVICES
,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=MODIFY
Changes the state of an exit routine.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that has been defined to the dynamic exits facility. If the name contains fewer than 16 characters, left-justify the name and pad the field with blanks.

,MODNAME=*modname*
Specifies an 8-byte field (or a register containing the address of an 8-byte field) that contains the 8-character name of the exit routine whose state you want to change. The first character must not be X'00' or blank. *modname* designates a load module or alias, whose entry point is the starting address of the exit routine.

,STATE=UNCHANGED
,STATE=ACTIVE
,STATE=INACTIVE
Specifies that you want the state of the exit routine to be unchanged or that you want to change the state to active or inactive.

- An active exit routine is associated with an exit and will be called when the exit is called.

- An inactive exit routine is associated with an exit, but will not be called when the exit is called.

,JOBNAME=*jobname*
,STOKEN=*stoken*
Specifies a condition under which the exit routine is to get control. You can require that the exit routine take control:

- During the execution of a specific job, or jobs
- While a specific address space is the primary address space.

JOBNAME specifies an area (or a register containing the address of an area) that contains the 8-character name of the job that must be running at the time the exit routine is to get control. To indicate the name of more than one job, use an asterisk for the last non-blank character. A matching jobname is one that matches all the characters preceding the asterisk.

To indicate that the exit routine is not to be restricted to a particular job, specify a jobname of C'* '. To leave the jobname unchanged, specify a jobname with the first character X'00' or blank. If STOKEN was specified when the module was added or modified, and the jobname does not indicate "unchanged," this jobname will be used instead.

Note: JOBNAME=ANY, which is value for REQUEST=ADD, is not valid for REQUEST=MODIFY.

STOKEN specifies an area (or a register containing the address of an area) that contains the 8-character STOKEN of the address space that must be the primary address space at the time the exit routine receives control. If JOBNAME was specified when the module was added, this STOKEN will be used instead.

,SERVICEMASK=*servicemask*
,SERVICEMASK=ALL_SERVICES
An optional input parameter containing the mask for this exit routine, to be applied to the service ID of the exit caller. Only if the ANDed value of the service mask and the service ID is nonzero for a given call and a given exit routine will that exit routine be called. The default is ALL_SERVICES.

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

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See "Return and reason codes" on page 407 for the return and reason codes for the CSVDYNEX MODIFY request.

Example

For an example of how to change the state of an exit routine, see "Example 2" on page 415.

Delete an exit routine from an exit

The CSVDYNEX DELETE request deletes an exit routine that was added by a CSVDYNEX ADD request to an exit that has been defined to the dynamic exits facility. On the CSVDYNEX DELETE request, you:

- Identify the exit (EXITNAME parameter) and the exit routine (MODNAME parameter)
- Tell the system whether or not to free the storage for the exit routine immediately (FORCE parameter).

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF UPDATE authority to FACILITY class entity CSVDYNEX.exitname.modname.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX DELETE request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX DELETE request.

Syntax

The standard form of the DELETE request on the CSVDYNEX macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
ƀ	One or more blanks must follow CSVDYNEX.
REQUEST=DELETE	
,EXITNAME=*exitname*	*exitname*: RS-type address or address in register (2) - (12).
,MODNAME=*modname*	*modname*: RS-type address or address in register (2) - (12).
,FORCE=NO	**Default:** FORCE=NO
,FORCE=YES	
,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=DELETE
Deletes an exit routine from an exit.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit associated with the exit routine you want to delete. If the name contains fewer than 16 characters, left-justify the name and pad the field with blanks.

,MODNAME=*modname*
Specifies an 8-byte field (or a register containing the address of an 8-byte field) that contains the 8-character name of the exit routine that you want to delete. The first character must not be X'00' or blank. Names of fewer than 8 characters must be left-justified and padded with blanks.

,FORCE=NO
,FORCE=YES
Indicates whether the system is to force the freeing of the storage for the exit routine when the routine is deleted. Specify FORCE=YES for an exit that has FASTPATH processing in effect, and either a PSW key 8 to 15 or ANYKEY processing in effect. For those exits, the system relies on you to tell it when to delete the storage. Assuming the exit has FASTPATH processing in effect, and the PSW key is 8 to 15 or ANYKEY processing is in effect:

- FORCE=NO, the default, tells the system to change the state of the exit routine to inactive. The system does not free the storage.
- FORCE=YES tells the system to free the storage of the exit routine immediately. Use FORCE=YES only if you are sure that this exit routine is not associated with any other exit.

For exits that are not FASTPATH or whose PSW key is 0 to 7 and that are not ANYKEY, the system frees the storage when no other exits are using the exit routine.

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 407 for the return and reason codes for the CSVDYNEX DELETE request.

Example

For an example of how to delete an exit routine from an exit, see “Example 9” on page 422.

Remove the definition of an exit

The CSVDYNEX UNDEFINE request removes the definition of an exit that was established by the DEFINE request. The EXITNAME parameter identifies the exit whose definition is to be removed.

Any exit routines associated with the exit whose definition is removed remain associated with the exit, and the exit is said to be implicitly defined.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF UPDATE authority to FACILITY class entity CSVDYNEX.exitname.UNDEFINE.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX UNDEFINE request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX UNDEFINE request.

Syntax

The standard form of the UNDEFINE request on the CSVDYNEX macro is written as follows:

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

Syntax	Description
CSVDYNEX	
ƀ	One or more blanks must follow CSVDYNEX.
REQUEST=UNDEFINE	
,EXITNAME=*exitname*	*exitname*: 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=UNDEFINE
Removes the definition of the exit. This exit was defined through a CSVDYNEX DEFINE request.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that has been defined to the dynamic exits facility. If the name has fewer than 16 characters, left-justify the name and pad the field with blanks.

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 407 for the return and reason codes for the CSVDYNEX UNDEFINE request.

Example

For an example of how to remove the definition of an exit, see “Example 10” on page 422.

Change the attributes of an exit

The CSVDYNEX ATTRIB request tells the system how to handle return code information that is returned from the call of multiple exit routines at an exit. On the RETINFO parameter of the CSVDYNEX CALL request, you tell the system what information to return to the issuer of the CALL request. Unless RETINFO=ALL was specified, you can use the CSVDYNEX ATTRIB request to override the RETINFO parameter specified on the CALL request.

On the CSVDYNEX ATTRIB request, you:

- Identify the exit (EXITNAME parameter)
- Identify how the system is to handle the return information from exit routines associated with the exit (KEEPRC, KEEPRCCOMP, and KEEPRCCVAL parameters).

If you use the CSVDYNEX ATTRIB request to specify return code processing for an exit that has not been defined, that exit is said to be implicitly defined.

Note: To control the handling of return code information from multiple exit routines, an installation can either write a program that uses CSVDYNEX REQUEST=ATTRIB, or use the SETPROG and SET PROG=xx commands.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF UPDATE authority to FACILITY class entity CSVDYNEX.exitname.ATTRIB.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX ATTRIB request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX ATTRIB request.

Syntax

The standard form of the ATTRIB request on the CSVDYNEX macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
ƀ	One or more blanks must follow CSVDYNEX.
REQUEST=ATTRIB	
,EXITNAME=*exitname*	*exitname*: RS-type address or register (2) - (12).
,KEEPRC=*keeprc*,KEEPRCCOMP=*option*	
,KEEPRC=*keeprc*,KEEPRCCOMP=VALUE,	
KEEPRCCVAL=*keeprccval*	
	keeprc: RS-type address or address in register (2) - (12).
	option: See the KEEPRCCOMP parameter description
	keeprccval: 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=ATTRIB
Changes the attributes of an exit.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that has been defined to the dynamic exits facility and whose attributes are to change. If the name contains fewer than 16 characters, left-justify the name and pad the field with blanks.

,KEEPRC=*keeprc*,KEEPRCCOMP=*option*

,KEEPRC=*keeprc*,KEEPRCCOMP=VALUE,KEEPRCCVAL=*keeprccval*
Specifies how the system is to process the return codes from exit routines associated with an exit. If RETINFO=ALL is specified on the CALL request, KEEPRC does not apply, because all information is returned. If RETINFO=LOWEST, HIGHEST, or LAST was specified on the CALL request, the KEEPRC parameter will override the RETINFO parameter.

KEEPRC=*keeprc* specifies a fullword (or a register containing the address of a fullword) that contains a value known as the KEEPRC return code. The system compares the return code from each exit routine called for this exit to the KEEPRC return code, using the comparison designated by KEEPRCCOMP. If the comparison is favorable, the system returns the values of registers 0, 1, and 15 to the issuer of the CSVDYNEX CALL request in the area specified on the RETAREA parameter.

If the comparison is not favorable, the system returns the values for another exit routine that is called, according to the values specified on the RETINFO parameter on the CSVDYNEX CALL request. It applies the rules specified on the RETINFO parameter unless it finds a match according to the rules specified on the KEEPRC parameter.

KEEPRCCOMP specifies the type of comparison the system makes between the KEEPRC value and the actual return code. The options on KEEPRCCOMP are:

- EQ - equal
- NE - not equal
- GT - greater than
- LT - less than
- GE - greater than or equal to
- LE - less than or equal to
- VALUE - the type of comparison is specified on the KEEPRCCVAL parameter.

KEEPRCCVAL=*keeprccval* specifies a one-byte field (or a register containing the address of a one-byte field) that contains the value that indicates the type of comparison the system is to do when comparing the actual return code of the exit routine with the value provided in KEEPRC. The constants produced by the list form of the macro can be used. For example, CSVDYNEX MF=(L,MYLIST) would produce such equate symbols as MYLIST_XKEEPRCCOMP_EQ and MYLIST_XKEEPRCCOMP_GT.

KEEPRCCVAL applies only to KEEPRCCOMP=VALUE, and is required with that parameter.

The default for the KEEPRC parameter and related parameters is that the system does no matching of the return code.

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See "Return and reason codes" on page 407 for the return and reason codes for the CSVDYNEX ATTRIB request.

Example

For an example of how to change the attributes of an exit, see "Example 10" on page 422.

List information about one or more exits

The CSVDYNEX LIST request returns information about all exits in the system that have been defined to the dynamic exits facility, or about specific exits (EXITNAME parameter). The information returned includes:

- The definition of the exit established on the DEFINE request, including:
  - Addressing mode
  - Reentrancy
  - Whether FASTPATH processing applies to the exit
  - Whether the exit has been explicitly or implicitly defined.
- Characteristics of the exit routines associated with the exit, as specified on the ADD request:
  - Name of the exit routine
  - State of the exit routine
  - Whether jobname filtering was requested
  - Whether STOKEN filtering was requested
  - The STOKEN and JOBNAME, if provided.

The system returns the information in an area you provide (ANSAREA and ANSLEN parameters), which is mapped by the CSVEXAA mapping macro.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF READ authority to FACILITY class entity CSVDYNEX.LIST.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit

Environmental factor	Requirement
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks may be held.
Control parameters:	Control parameters and the area where the system places the information obtained through the CSVDYNEX LIST request (ANSAREA parameter) must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and associated values for return and reason codes returned by CSVDYNEX. Also include the CSVEXAA mapping macro to get a mapping of the output area specified by the ANSAREA parameter. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX LIST request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX LIST request.

Syntax

The standard form of the LIST request on the CSVDYNEX macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
REQUEST=LIST	

Syntax	Description
,EXITNAME=ALL_EXITS	**Default:** EXITNAME=ALL_EXITS
,EXITNAME=*exitname*	*exitname*: 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).
,EXAAVER=0	**Default:** EXAAVER=0
,EXAAVER=1	
,EXAAVER=2	
,EXAAVER=3	
,EXITTYPE=ANY	**Default:** EXITTYPE=ANY
,EXITTYPE=INSTALLATION	
,EXITTYPE=PROGRAM	
,EXITTYPE=NOTPROGRAM	
,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=LIST
Lists information about one or more exits.

,EXITNAME=ALL_EXITS

,EXITNAME=*exitname*
EXITNAME=ALL_EXITS, the default, tells the system to list information about all exits in the system that have been defined to the dynamic exits facility.

EXITNAME=*exitname* specifies a 16-byte field (or a register containing the address of a 16-byte field) that contains the 16-character name of the exit. If the name contains fewer than 16 characters, left-justify the name and pad the field with blanks. To indicate the name of more than one exit, use an asterisk for the last non-blank character. A matching exit name is one that matches all the characters preceding the asterisk. If the first character of the name is X'00', the system processes the request as if EXITNAME=ALL_EXITS has been specified.

,ANSAREA=*ansarea*
This parameter specifies an area (or a register containing the address of an area) where the system is to store information associated with the exits. Use the CSVEXAA mapping macro to map this area.

(See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).) Specify the length of this area on the ANSLEN parameter.

,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 list. This value must be equal to or greater than the length of the EXAAHDR structure in the CSVEXAA mapping macro.

If the area is not long enough to contain all the information, the system returns as many entries as it can. The system returns the length that is currently required to contain all the information in the EXAAHTLEN field in the CSVEXAA mapping macro, with return code CSVDYNEXRC_WARN (4) and reason code CSVDYNEXRSNNOTALLDATARETURNED (X'xxxx0403').

,EXAAVER=0
,EXAAVER=1
,EXAAVER=2
,EXAAVER=3
Specifies the format of information to be returned, as mapped by DSECTs within the CSVEXAA data area. EXAAVER=3 returns more information about exit routines than EXAAVER=2, which returns more information than EXAAVER=1, which returns more information than EXAAVER=0. All of these return header information mapped by DSECT EXAAHDR and exit information mapped by DSECT EXAAE. The default is EXAAVER=0.

- EXAAVER=0, the default, specifies that exit routine information is mapped by the EXAAM DSECT.
- EXAAVER=1 specifies that exit routine information is mapped by the EXAAM1 DSECT.
- EXAAVER=2 specifies that exit routine information is mapped by the EXAAM2 DSECT.
- EXAAVER=3 specifies that exit routine information is mapped by the EXAAM3 DSECT.

,EXITTYPE=ANY
,EXITTYPE=INSTALLATION
,EXITTYPE=PROGRAM
,EXITTYPE=NOTPROGRAM
When REQUEST=LIST is specified, EXITTYPE is an optional parameter that indicates the type of exit about which to return data.

The default is EXITTYPE=ANY

EXITTYPE=ANY indicates that data is returned for any exit type.

EXITTYPE=INSTALLATION indicates that data is returned only if the exit is marked as an installation exit.

EXITTYPE=PROGRAM indicates that data is returned only if the exit is marked as a program exit.

EXITTYPE=NOTPROGRAM indicates that data is returned only if the exit is not marked as a program exit.

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 407 for the return and reason codes for the CSVDYNEX LIST request.

Example

For an example of how to list information about exits, see “Example 6” on page 420.

Call one or more exit routines at an exit

The CSVDYNEX CALL request passes control to the exit routine or routines associated with an exit that has been explicitly defined to the dynamic exits facility (that is, that has been defined using the CSVDYNEX DEFINE request). The caller can:

- Specify the information that the exit routine is to find in certain GPRs at entry
- Receive information from the exit routines in the return area (RETAREA and RETLEN parameters)
- Receive a token that identifies an exit routine that did not get control at the exit (NEXTTOKEN).

Additionally, on the CSVDYNEX CALL request you can:

- Specify how the system is to handle the return codes from more than one exit routine (RETINFO parameter)
- Specify that the exit is to have FASTPATH processing (FASTPATH parameter).

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	For CALL FASTPATH=YES requests, problem state and any PSW key. For CALL FASTPATH=NO requests, any of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized (not allowed in cross memory mode) • SAF READ authority to FACILITY class entity CSVDYNEX.exitname.CALL (not allowed in cross memory mode)
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	For CALL with FASTPATH=YES requests, 31-bit. For all other requests, 24- or 31-bit.
ASC mode:	For CALL,FASTPATH=YES, primary mode. For all other requests, primary or AR mode.
Interrupt status:	Enabled for I/O and external interrupts. When an exit is defined with DISABLEDCALL=OK, CSVDYNEX REQUEST=CALL,FASTPATH=YES can be used when disabled for I/O and external interrupts.
Locks:	The local and CMS locks may be held, but are not required.

Environmental factor	Requirement
Control parameters:	Control parameters and the return area (through RETAREA) must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). When CSVDYNEX REQUEST=CALL is used while disabled for I/O and external interrupts, all user-provided storage areas (such as the parameter list, RUB area, RETAREA, NEXTTOKEN area, WORKAREA) must be page-fixed or DREF (disabled-reference).

Programming requirements

Include the CSVEXRET mapping macro to get a mapping of the area provided through the RETAREA parameter. This macro also defines variables and values for return and reason codes returned by CSVDYNEX. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

Restrictions

1. The FASTPATH parameter is not valid on the modify form of CSVDYNEX REQUEST=CALL.
2. If the caller uses FASTPATH=NO and is in cross memory mode, the exit routine gets control with the secondary address space equal to the primary address space of the caller.
3. If the caller uses FASTPATH=NO and is in problem state, the caller must not have SPIE or ESPIE routines in effect.
4. FASTPATH=YES is valid on the CSVDYNEX CALL request only if the CSVDYNEX DEFINE request was issued with FASTPATH=YES.
5. The PSW key of the caller must be the same as the value specified on the KEY parameter of the DEFINE request, or must be 0.

Input register information

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX CALL request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX CALL request.

Syntax

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

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

Syntax	Description
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
REQUEST=CALL	
,EXITNAME=*exitname*	*exitname*: RS-type address or address in register (2) - (12).
,RUB=*rub*	*rub*: RS-type address or address in register (2) - (12).
,RETAREA=*retarea*	*retarea*: RS-type address or address in register (2) - (12).
,RETLEN=*retlen*	*retlen*: RS-type address or address in register (2) - (12).
,RETINFO=LAST	**Default:** RETINFO=LAST
,RETINFO=LOWEST	
,RETINFO=HIGHEST	
,RETINFO=ALL	
,NEXTTOKEN=*nexttoken*	*nexttoken*: RS-type address or address in register (2) - (12).
,FASTPATH=NO	**Default:** FASTPATH=NO
,FASTPATH=YES,WORKAREA=*workarea*	
	workarea: RS-type address or address in register (2) - (12).
,EXRETVER=0	**Default:** EXRETVER=0
,EXRETVER=1	
,SERVICEID=*serviceid*	*serviceid*: RS-type address or address in register (2)-(12)

Syntax	Description
,SERVICEID=NO_SERVICEID	**Default:** SERVICEID=NO_SERVICEID
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0 - 1
,MF=S	

Parameters

The parameters are explained as follows:

REQUEST=CALL
Calls one or more exit routines at an exit.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that is to invoke one or more exit routines. If the name contains fewer than 16 characters, left-justify the name and pad the field with blanks.

,RUB=*rub*
Specifies an area (or a register containing the address of an area) that is known as the register update block (RUB). The RUB is a list of fullwords that contains information to be placed in certain GPRs at entry to each exit routine that gets control at an exit point. You must obtain storage for the RUB, and initialize it as follows:

- The first two bytes in the first word identify which registers the system is to load with the data; GPR 0 corresponds to bit 0, GPR 1 corresponds to bit 1, and so on. The last two bytes should be 0. You indicate the registers by setting the corresponding bits. For example, if you want the data in GPRs 0, 1, and 13, the first word is as follows:
  - X'C0040000'
- The remaining words in the RUB contain the data that is to be in the registers, in the order 0 through 15. To continue the example, the rest of the list would be:
  - Second word: *xxxx* — value in register 0
  - Third word: *yyyy* — value in register 1
  - Fourth word: *zzzz* — value in register 13.

Processing is most efficient when word 0 contains one of the following:

- X'40040000' - indicating registers 1 and 13, or
- X'FFFC0000' - indicating registers 0 through 13.

The RUB area need only be long enough to contain the necessary information.

If the exit is defined as AMODE=24 or AMODE=UNDEFINED, do not specify GPR2 in the RUB.

,RETAREA=*retarea*
Specifies a field (or a register containing the address of a field) where the system returns information from the exit routines that are called. The mapping macro CSVEXRET maps this area. (See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary).)

,RETLEN=*retlen*
Specifies a fullword field (or a register containing the address of a fullword field) that tells the system the length of the RETAREA. The RETAREA must be long enough to contain at least one return entry. The size of the RETAREA for *n* entries is “L'EXRETHDR+n*L'EXRETINFO”.

,RETINFO=LAST
,RETINFO=LOWEST
,RETINFO=HIGHEST
,RETINFO=ALL
Tells the system how to handle return code information when the exit calls more than one exit routine.

- RETINFO=LAST, the default, indicates that the return area contains the information from the last exit routine called. If the last exit routine abnormally ended, its information is placed in the return area only if it was the only exit routine called. If it was not the only exit routine called, the information from the most recent exit routine that did not end abnormally is returned.
- RETINFO=LOWEST indicates that the return area contains the information from the exit routine whose GPR15 return value was the lowest. (The system considers the contents of GPR15 as a 4-byte unsigned quantity for the purposes of this calculation.) If the exit routine abnormally ends, its information is placed in the return area only if it was the first exit routine called. That information will be overlaid by any other exit routine's return information. If multiple exit routines return with the same lowest value, only the return information from the first routine will be returned.
- RETINFO=HIGHEST indicates that the return area contains the information from the exit routine for which the GPR15 return value was the greatest. (The system considers the contents of GPR15 as a 4-byte unsigned quantity for the purpose of this calculation.) If the exit routine abnormally ends, its information is placed into the return area only if it was the first exit routine called. That information will be overlaid by any other exit routine's return information. If multiple exit routines return with the same highest value, the return information from the first will be returned.
- RETINFO=ALL indicates that information from each exit routine is returned. If the return area fills completely, the system stops calling exit routines at that exit and control returns to the caller for analysis. To invoke the remaining exit routines, the caller can reissue the CSVDYNEX CALL request, passing the NEXTTOKEN value that the system returned.

,NEXTTOKEN=*nexttoken*
Specifies a field (or a register containing the address of a field) that is both input and output. The contents are an 8-character token that signifies whether the exit routine has been called.

- If this is the first CALL request for this iteration of the exit, place zeroes in the field.
- If the field contains a value other than zeroes, the token in the field signifies that more exit routines remain to be processed at this exit. The system stores the token in the field during CSVDYNEX CALL processing, when there is not enough storage in RETAREA to hold all the information; or during CSVDYNEX RECOVER processing.

,FASTPATH=NO
,FASTPATH=YES,WORKAREA=*workarea*
Specifies whether FASTPATH processing will occur for this CALL request. On the CSVDYNEX DEFINE request, the FASTPATH keyword enables the FASTPATH function. On the CSVDYNEX CALL request, it specifies whether or not to use the function.

- FASTPATH=NO, the default, specifies that FASTPATH processing is not to occur for this CALL request.

- FASTPATH=YES specifies that FASTPATH processing is to occur for this CALL request. FASTPATH processing requires that you provide a recovery routine to handle recovery for CSVDYNEX CALL processing, and that you issue the CSVDYNEX RECOVER request within that recovery routine.

  WORKAREA=*workarea* specifies a 512-character field (or a register containing the address of a field) that is used by the system to handle recovery of the exit routine. The WORKAREA parameter is required for FASTPATH processing. The rules for specifying WORKAREA are:

  – The work area should be aligned on a doubleword boundary.
  – Before you issue the first CSVDYNEX CALL request, zero the first word of the work area.

,EXRETVER=0
,EXRETVER=1
Specifies the format of information to be returned, as mapped by the DSECTs within the CSVEXRET data area. EXRETVER=1 returns more information about exit routines than EXRETVER=0.

- EXRETVER=0, the default, specifies that return information is mapped by the EXRET DSECT.
- EXRETVER=1 specifies that return information is mapped by the EXRET1 DSECT.

,SERVICEID=*serviceid*
,SERVICEID=NO_SERVICEID
An optional input parameter containing the identifier of the particular service being called. This 8-byte value is ANDed with the ServiceMask of the eligible exit routines. Only exit routines for which the ANDed value is nonzero are called. The default is NO_SERVICEID.

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

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,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** indicates the original parameter list version of the macro.

1
Version **1** introduces the following parameter:

- SERVICEID

To code: Specify in this input parameter one of the following values:

- IMPLIED_VERSION
- MAX
- A decimal value in the range of 0 - 1

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

The CSVDYNEX CALL request with FASTPATH=YES might abnormally terminate with abend code X'0C4' if you provide a field that is not accessible. See *z/OS MVS System Codes* for an explanation and programmer response.

Return and reason codes

See "Return and reason codes" on page 407 for the return and reason codes for the CSVDYNEX CALL request.

Example

See three examples of calling exit routines at an exit, in "Example 3" on page 415, "Example 4" on page 416, and "Example 5" on page 417.

Provide recovery for an exit routine that abnormally ended

You use the CSVDYNEX RECOVER request within the recovery routine of the program that issues a CSVDYNEX CALL request with FASTPATH processing. This request is required for FASTPATH processing; without it, the system does not get back control from the abnormally ending exit routine.

On the CSVDYNEX RECOVER request you:

- Identify the exit (EXITNAME parameter).
- Provide a work area for the system to use (WORKAREA parameter).
- Provide an area where the system returns information needed when not all the exit routines at the exit point have gotten control at the time of recovery processing. This allows processing to continue with the next routine (NEXTTOKEN parameter).
- Provide an area where the system places information from the exit routine that abnormally ended (RETAREA and RETLEN parameters).
- Give the system the address of the SDWA (SDWA parameter).

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Any of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized (not allowed for callers in cross memory mode) • SAF READ authority to FACILITY class entity CSVDYNEX.exitname.RECOVER (not allowed for callers in cross memory mode)
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
Locks:	Local and CMS locks may be held, but are not required.
Control parameters:	Control parameters and the return area (through RETAREA) must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to get a mapping of the area provided through the RETAREA parameter. This macro also defines variables and values for return and reason codes returned by CSVDYNEX. (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

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX RECOVER request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX RECOVER request.

Syntax

The standard form of the RECOVER request on the CSVDYNEX macro is written as follows:

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

Syntax	Description
ƀ	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
ƀ	One or more blanks must follow CSVDYNEX.
REQUEST=RECOVER	
,EXITNAME=*exitname*	*exitname*: RS-type address or address in register (2) - (12).
,WORKAREA=*workarea*	*workarea*: RS-type address or address in register (2) - (12).
,NEXTTOKEN=*nexttoken*	*nexttoken*: RS-type address or address in register (2) - (12).
,RETAREA=*retarea*	*retarea*: RS-type address or address in register (2) - (12).
,RETLEN=*retlen*	*retlen*: RS-type address or address in register (2) - (12).
,SDWA=*sdwa*	*sdwa*: RS-type address or address in register (2) - (12).
,EXRETVER=0	**Default:** EXRETVER=0
,EXRETVER=1	
,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=RECOVER
Provides recovery for an exit routine that abnormally ended.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that has been defined to the dynamic exits facility. If the name contains fewer than 16 characters, left-justify the name and pad the field with blanks.

,WORKAREA=*workarea*
Specifies a 512-character field (or a register containing the address of a field) that the system uses while providing recovery for the exit routine. Align this area on a doubleword boundary. The WORKAREA specified on the RECOVER request must be the WORKAREA that was passed on the CALL request.

,NEXTTOKEN=*nexttoken*
Specifies a field (or a register containing the address of a field) where the system places an 8-character token that identifies an exit routine that did not get control. The next issuer of the CALL request passes it to the system through the NEXTTOKEN= parameter of the CSVDYNEX CALL request. The NEXTTOKEN specified on the next CALL request should be the NEXTTOKEN specified on the RECOVER request.

,RETAREA=*retarea*
Specifies a field (or a register containing the address of a field) where the system tells the caller where the error occurred. The mapping macro CSVEXRET maps this area.

,RETLEN=*retlen*
Specifies a field (or a register containing the address of a field) that tells the system the length of RETAREA. The RETAREA must be long enough to contain at least one entry. The size of the RETAREA for *n* entries is “L'EXRETHDR+n*L'EXRETINFO”.

,SDWA=*sdwa*
Specifies a fullword address (or a register containing the address) of the SDWA associated with an abnormal ending of an exit routine (found in GPR 1 on entry to the recovery routine). If no SDWA was passed to the recovery routine, set the fullword to zero.

,EXRETVER=0
,EXRETVER=1
Specifies the format of information to be returned, as mapped by DSECTs within the CSVEXRET data area. EXRETVER=1 returns more information about exit routines than EXRETVER=0.

- EXRETVER=0, the default, specifies that return information is mapped by the EXRET DSECT.
- EXRETVER=1 specifies that return information is mapped by the EXRET1 DSECT.

,RETCODE=*retcode*
Specifies a fullword (or a register containing the address of a fullword) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register containing the address of a fullword) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 407 for the return and reason codes for the CSVDYNEX RECOVER request.

Example

For an example of how to provide recovery for an exit routine that has abnormally ended, see “Example 5” on page 417.

Determine whether an exit routine exists for an exit

The CSVDYNEX QUERY request returns information in a return code that tells you whether there are any exit routines associated with an exit. You can then decide whether to proceed with a CSVDYNEX CALL or ADD request (QTYPE=CALL or QTYPE=ADD parameter). Examples of the use of this request are:

- If no exit routines are associated with the exit, you can omit the CSVDYNEX CALL request.
- If no exit routines are associated with the exit through a parmlib member or a SETPROG command, you can use the CSVDYNEX ADD request to add a default exit routine.

If the exit has been defined implicitly, you will receive a warning return and reason code.

On the CSVDYNEX QUERY request you identify the exit (EXITNAME parameter) and provide an area for the system to use (WORKAREA parameter).

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:	31-bit
ASC mode:	Primary
Interrupt status:	Enabled for I/O and external interrupts. When an exit is defined with DISABLEDCALL=OK, CSVDYNEX REQUEST=QUERY can be used when disabled for I/O and external interrupts.
Locks:	The local and CMS locks may be held, but are not required.
Control parameters:	Control parameters must be in the primary address space. When CSVDYNEX REQUEST=QUERY is used while disabled for I/O and external interrupts, control parameters must be page-fixed or DREF (disabled-reference).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. (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

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX QUERY request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX QUERY request.

Syntax

The standard form of the QUERY request on the CSVDYNEX macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
REQUEST=QUERY	
,EXITNAME=*exitname*	*exitname*: RS-type address or address in register (2) - (12).
,WORKAREA=*workarea*	*workarea*: RS-type address or address in register (2) - (12).
,QTYPE=ADD	
,QTYPE=CALL	**Default:** QTYPE=CALL
,ADDRSPACE=PRIMARY	**Default:** ADDRSPACE=PRIMARY
,ADDRSPACE=ANY	
,SERVICEID=*serviceid*	*serviceid*: RS-type address or address in register (2)-(12)
,SERVICEID=NO_SERVICEID	**Default:** SERVICEID=NO_SERVICEID
,RETCODE=*retcode*	*retcode*: RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0 -1
,MF=S	

Parameters

The parameters are explained as follows:

REQUEST=QUERY
Asks whether an exit routine exists for an exit.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that has been defined to the dynamic exits facility. The characters must be left-justified in a 16-character field and padded with blanks.

,WORKAREA=*workarea*
Specifies a field (or a register containing the address of a field) that provides the 512-character work area that the system uses. Align this area on a doubleword boundary.

,QTYPE=ADD
,QTYPE=CALL
Indicates the type of QUERY request:

- QTYPE=ADD is issued before a CSVDYNEX ADD request. A return code of X'00' indicates that at least one exit routine is associated with the exit. This return code is issued for QTYPE=ADD even if all associated exit routines are inactive, no matter the reason for the inactive state.
- QTYPE=CALL, the default, is issued before a CSVDYNEX CALL request. A return code of X'00' indicates that an active exit routine was added. The current JOBNAME and STOKEN of this exit routine match the JOBNAME and STOKEN criteria specified when the exit routine was associated with the exit. Note that it is not necessary to use this prior to using CSVDYNEX=CALL. If you issue a CSVDYNEX CALL request and there are no exit routines associated with the exit, the system returns with return code CSVDYNEXRC_WARN (4) and reason code CSVDYNEXRSNNOMODULES (X'xxxx0406').

  ,ADDRSPACE=PRIMARY
  ,ADDRSPACE=ANY
  When QTYPE=CALL is specified, an optional parameter that indicates the address space to check with respect to a subsequent REQUEST=CALL. The default is ADDRSPACE=PRIMARY.

  – ADDRSPACE=PRIMARY indicates to check for the case where the QTYPE=CALL is for the primary address space.
  – ADDRSPACE=ANY indicates to check for the case where the QTYPE=CALL could be for any address space. An active exit routine added with STOKEN or JOBNAME will match if this is specified, even if the primary address space does not match the STOKEN/JOBNAME.

,SERVICEID=*serviceid*
,SERVICEID=NO_SERVICEID
When QTYPE=CALL and REQUEST=QUERY are specified, an optional input parameter containing the identifier of the particular service being called. This 8-byte value is ANDed with the ServiceMask of the eligible exit routines. Only exit routines for which the ANDed value is nonzero are called. The default is NO_SERVICEID.

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

,RETCODE=*retcode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,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** indicates the original parameter list version of the macro.

1
Version **1** introduces the following parameter:

- SERVICEID

To code: Specify in this input parameter one of the following values:

- IMPLIED_VERSION
- MAX
- A decimal value in the range of 0 - 1

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

The CSVDYNEX QUERY request might abnormally terminate with abend code X'0C4' if you provide a field that is not accessible. See *z/OS MVS System Codes* for an explanation and programmer response.

Return and reason codes

See “Return and reason codes” on page 407 for the return and reason codes for the CSVDYNEX QUERY request.

Example

For an example of how to identify whether an exit routine exists for an exit, see “Example 7” on page 421.

Replace an exit routine for an exit

Replacing an exit routine for an exit means removing the association of an active exit routine and replacing it with a newer version while making sure that either the old or new version, but not both, gets control when the exit is called.

On the CSVDYNEX REPLACE request, you:

- Name the exit that the exit routine is to be associated with (EXITNAME parameter)
- Identify where to find the exit routine (MODNAME, DSNAME, and MODADDR parameters)
- Request that the system send a message to the operator if the system encounters certain error conditions when processing the exit routine (MESSAGE parameter)

The CSVDYNEX DELETE request deletes an exit routine that was added to an exit by the CSVDYNEX REPLACE request.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	One of the following: • Supervisor state • PSW key 0-7 • PKM allowing key 0-7 • APF-authorized • SAF UPDATE authority to FACILITY class entity CSVDYNEX.exitname.modname.
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks to be held.
Control parameters:	Control parameters and the area that contains the data set name (specified on DSNAME) must be in the primary address space or, for AR-mode callers, must be in an address space or data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL).

Programming requirements

Include the CSVEXRET mapping macro to define symbolic names and values for return and reason codes returned by CSVDYNEX. See *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for more information.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

See “Input register information for CSVDYNEX” on page 358 for input register information for the CSVDYNEX REPLACE request.

Output register information

See “Output register information for CSVDYNEX” on page 358 for output register information for the CSVDYNEX REPLACE request.

Syntax

The standard form of the REPLACE request on the CSVDYNEX macro is written as follows:

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
REQUEST=REPLACE	
,EXITNAME=*exitname*	*exitname*: RS-type address or register (2) - (12).
,MODNAME=*modname*	*modname*: RS-type address or register (2) - (12).
,MESSAGE=NO	**Default:** MESSAGE=NO
,MESSAGE=ERROR	
,MESSAGE=FOUNDBUTERROR	
,DSNAME=*dsname*	*dsname*: RS-type address or register (2) - (12).
,MODADDR=*modaddr*	*modaddr*: RS-type address or register (2) - (12).
,STATE=UNCHANGED	
,STATE=ACTIVE	
,STATE=INACTIVE	
,SERVICEMASK=*servicemask*	*servicemask*: RS-type address or address in register (2)-(12)
,SERVICEMASK=ALL_SERVICES	**Default:** SERVICEMASK=ALL_SERVICES
,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=REPLACE
Replaces an exit routine for an exit.

The following attributes of the added exit routine are maintained:

- the state (active or inactive)
- the jobname or STOKEN
- the ADDABENDNUM characteristic
- the ABENDCONSEC characteristic

The exit routine gets control in the same sequence of exit routines that the replaced routine did.

,EXITNAME=*exitname*
Specifies a 16-byte field (or a register containing the address of a 16-byte field) containing the 16-character name of an exit that has been defined to the dynamic exits facility.

If the name has fewer than 16 characters, you must left-justify the name and pad the field with blanks.

,MODNAME=*modname*
Specifies an 8-byte field (or a register containing the address of an 8-byte field) containing the name of an exit routine to be replaced for the exit.

The first character must not be X'00' or blank.

If you specify MODADDR, *modname* is the name of the exit routine.

If you do not specify MODADDR, *modname* designates a load module or alias. The entry point of the load module or alias is the starting address of the exit routine.

If the name has fewer than 8 characters, you must left-justify the name and pad the field with blanks.

,MESSAGE=NO
,MESSAGE=ERROR
,MESSAGE=FOUNDBUTERROR
Specifies whether the system is to send message CSV431I to the operator if the system encounters certain errors when processing the exit routine.

- **MESSAGE=NO**, the default, requests that the system send the return and reason codes only.
- **MESSAGE=ERROR** requests that the system send message CSV431I if any of the following situations are encountered:
  - The exit requires reentrancy; the exit routine is not reentrant.
  - The exit requires AMODE=31, but the exit routine is AMODE=24; or the exit requires AMODE=24, but the exit routine is AMODE=31.
  - The exit routine could not be located.
  - A CSVDYNEX REPLACE request specified that the exit routine must be loaded from a particular data set, but that data set is not APF-authorized, and the caller is in problem state, with PSW key 8-15, and is not APF-authorized.
- **MESSAGE=FOUNDBUTERROR** requests that the system send message CSV431I if it encounters any of the circumstances covered by MESSAGE=ERROR, except the following circumstance:
  - The exit routine could not be located.

,DSNAME=*dsname*
,MODADDR=*modaddr*
Tells the system how to find the exit routine to be added.

If neither DSNAME nor MODADDR are specified, the system attempts to find the module using LPA, the LNKLST concatenation, and the nucleus.

DSNAME specifies a field (or a register containing the address of a field) containing the 44-character name of a data set or library from which the module is to be obtained. The following actions must be performed when specifying DSNAME:

- Allocate the data set as a PDS or a PDSE.
- If the 44-character name has fewer than 44 characters, you must left-justify the name and pad it with blanks.
- Do not specify the data set name beginning with a blank or X'00'. The system responds as if you have specified no data set.
- If the caller is in problem state with PSW key 8 to 15, and is not APF-authorized, the data set must be APF-authorized.
- The data set must be cataloged.

Note:

1. Specify DSNAME only if dynamic allocations are enabled within the caller's primary address space.
2. If the data set has been migrated, your program waits for the system to retrieve it.

MODADDR specifies a fullword (or a register containing the address of a fullword) that contains the address of the exit routine to be added.

If the exit routine is to get control in 31-bit mode, bit 0 must be on. If the exit routine is to get control in 24-bit mode, bit 0 must be off.

The system assumes that the designated exit routine is reentrant.

MODADDR cannot be used if the caller is in problem state with PSW key 8 to 15 and is not APF-authorized.

If you specify MODADDR, ensure that the subpool and the key where the exit routine is stored are appropriate for the address spaces and keys in which the exit routine can get control. For example, if you specify STOKEN, the exit routine can be stored in the private area of the address space designated by STOKEN. If you do not specify STOKEN, and the exit routine can be called from other address spaces, ensure that the exit routine is stored in the common area.

If the storage is fetch-protected, the storage key must not conflict with the PSW key on entry to the exit routine. To prevent modification by unauthorized users, the storage for exit routines that get control in system key must not be PSW key 8-15.

,STATE=UNCHANGED
,STATE=ACTIVE
,STATE=INACTIVE

Specifies that you want the state of the exit routine to be unchanged or that you want to change the state to active or inactive.

An active exit routine is associated with an exit and is called when the exit is called.

An inactive exit routine is associated with an exit, but is not called when the exit is called.

,SERVICEMASK=*servicemask*
,SERVICEMASK=ALL_SERVICES

An optional input parameter containing the mask for this exit routine, to be applied to the service ID of the exit caller. Only if the ANDed value of the service mask and the service ID is nonzero for a given call and a given exit routine will that exit routine be called. The default is ALL_SERVICES.

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

,RETCODE=*retcode*

Specifies a fullword (or a register) where the system is to store the CSVDYNEX return code. The return code is also in GPR 15.

,RSNCODE=*rsncode*
Specifies a fullword (or a register) where the system is to store the CSVDYNEX reason code. The reason code is also in GPR 0.

,MF=S
Specifies the standard form of the CSVDYNEX macro.

ABEND codes

None.

Return and reason codes

See "Return and reason codes" on page 407 for the return and reason codes for the CSVDYNEX REPLACE request.

Example

None.

Return and reason codes

When the CSVDYNEX macro returns control to your program, GPR 15 (and *retcode*, if you coded RETCODE) contains a return code. When the value in GPR 15 is not zero, GPR 0 (and *rsncode*, if you coded RSNCODE) contains a reason code.

Macro CSVEXRET provides equate symbols for the return and reason codes. The equate symbols associated with each return code are as follows:

0
CSVDYNEXRC_OK

4
CSVDYNEXRC_WARN

8
CSVDYNEXRC_INVPARM

C
CSVDYNEXRC_ENV

10
CSVDYNEXRC_COMPERROR

The following table identifies the hexadecimal return and reason codes and the equate symbol associated with each reason code. *xxxx* indicates information that you might need to provide to IBM support personnel.

Table 47. Return and Reason Codes for the CSVDYNEX Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
00	—	**Meaning**: The CSVDYNEX request completed successfully. The result depends on the request: • DEFINE - An exit is defined. • ADD - An exit routine is added. • MODIFY - An exit routine is modified. • REPLACE - An exit routine is replaced. • DELETE - An exit routine is deleted. • UNDEFINE - The definition of an exit is removed. • ATTRIB - Attributes of an exit are changed. • LIST - Information about exit routines is listed. • CALL - An exit routine or exit routines are called. • RECOVER - Recovery for an exit routine is completed; there are no more exit routines to call. • QUERY - For QTYPE=ADD, an exit routine that was added through a PROG=xx system parameter or a SETPROG or SET PROG=xx operator command is associated with the exit. For QTYPE=CALL, an active exit routine is associated with the exit. The current JOBNAME and STOKEN of this exit routine match the JOBNAME and STOKEN criteria specified when the exit routine was associated with the exit. **Action**: None.
04	xxxx0401	CSVDYNEXRSNALREADYEXISTS **Meaning**: The request completed successfully. The result depends on the request: • For ADD: the exit routine was already associated with the exit. • For DEFINE: the exit already exists. **Action**: Make sure you specified the correct exit or exit routine name.
04	xxxx0402	CSVDYNEXRSNDOESNOTEXIST **Meaning**: One of the following: • For DELETE, MODIFY, or REPLACE: the exit routine is not associated with the exit. • For UNDEFINE: the exit is not defined. **Action**: Make sure you specified the correct exit or exit routine name.
04	xxxx0403	CSVDYNEXRSNNOTALLDATARETURNED **Meaning**: For LIST: not all the data was returned because the answer area is not large enough. **Action**: Check the answer area field EXAAHTLEN in the CSVEXAA mapping macro to see how much space is required to return the information. Expand the ANSAREA to hold all the information. Issue the CSVDYNEX macro again, specifying, on the ANSLEN parameter, a fullword containing a value large enough to contain the entire answer area.
04	xxxx0406	CSVDYNEXRSNNOMODULES **Meaning**: For CALL: no active exit routines associated with the exit match the current JOBNAME or STOKEN; that is, no exit routines were invoked. For QUERY (QTYPE=ADD), no exit routines (active or inactive) are currently associated with the specified exit through a PROG=xx parameter, SETPROG command, SET PROG=xx command, or CSVDYNEX REQUEST=ADD macro. For QUERY (QTYPE=CALL), no active exit routines associated with the exit match the current JOBNAME or STOKEN; that is, no routines will be invoked by CSYDYNEX REQUEST=CALL. **Action**: None.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
04	xxxx0407	CSVDYNEXRSNMOREMODULES **Meaning**: One of the following: • For a CALL request that specified RETINFO=ALL: there are more exit routines to call. • For a RECOVER request: there are more exit routines to call. **Action**: If you want the rest of the exit routines to be called for this exit, issue the CALL request again, specifying the NEXTTOKEN value returned from this request.
04	xxxx0408	CSVDYNEXRSNUSERKEYDELETENOFORCE **Meaning**: A DELETE request was made for an exit that was defined to be PSW key 8 or higher or ANYKEY=YES with FASTPATH=YES. The request did not specify FORCE=YES. The system changes the state of the exit routine to inactive and does not free the storage for that module. **Action**: If you are sure that the exit routine can be deleted from the system, reissue the DELETE request, specifying FORCE=YES.
04	xxxx0409	CSVDYNEXRSNQUERYNOTFOUND **Meaning**: A QUERY request was made for an exit that has not been defined. **Action**: Make sure you specified the correct exit name.
04	xxxx040A	CSVDYNEXRSNIMPLICITLYDEFINED **Meaning**: A QUERY request was made for an exit that has been defined implicitly rather than explicitly. You define an exit implicitly when: • You add exit routines to an exit before the exit has been defined • You set attributes for an exit using the ATTRIB request before the exit has been defined. **Action**: Make sure you specified the correct exit name.
08	xxxx0801	CSVDYNEXRSNBADPARMLIST **Meaning**: Program error. The system is unable to access the parameter list. **Action**: Make sure the parameter list you passed is in the correct PSW key.
08	xxxx0802	CSVDYNEXRSNSRBMODE **Meaning**: Program error. A program running in SRB mode entered a request that required task mode. **Action**: For the specified request, do not issue the CSVDYNEX macro while running in SRB mode.
08	xxxx0803	CSVDYNEXRSNNOTENABLED **Meaning**: Program error. A program issued the CSVDYNEX macro while running disabled for I/O or external interrupts. **Action**: Issue the CSVDYNEX macro while running enabled for I/O or external interrupts.
08	xxxx0804	CSVDYNEXRSNNOTAUTHORIZED **Meaning**: Program error. The caller is not authorized to issue the CSVDYNEX macro for the specified request. **Action**: See the authorization requirements described in the standard syntax for the specific request you issued.
08	xxxx0805	CSVDYNEXRSNHOMENOTPRIMARY **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 CSVDYNEX macro while running in cross memory mode.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0806	CSVDYNEXRSNBADANSAREAALET **Meaning**: Program error. For LIST: the ALET of the area specified on the ANSAREA parameter is incorrect. **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	CSVDYNEXRSNBADANSAREA **Meaning**: Program error. For LIST: 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	CSVDYNEXRSNBADANSLEN **Meaning**: Program error. For a LIST request: the length of the answer area specified on the ANSLEN parameter is not equal to or greater than the length of the EXAAHDR structure in the CSVEXAA mapping macro. **Action**: Expand the ANSAREA to a size large enough to contain the information. On the ANSLEN parameter, specify a fullword containing a value that is equal to or greater than the length of the EXAAHDR structure in the CSVEXAA mapping macro.
08	xxxx0809	CSVDYNEXRSNBADREQUESTTYPE **Meaning**: Program error. The system found an incorrect request type in the parameter list created by the CSVDYNEX macro. **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 you specified the COMPLETE option on at least one invocation.
08	xxxx080A	CSVDYNEXRSNBADESTAE **Meaning**: Program error. The CSVDYNEX 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	CSVDYNEXRSNRESERVEDNOT0 **Meaning**: Program error. The system found a non-zero reserved field in the parameter list that the CSVDYNEX 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 you specified the COMPLETE option on at least one invocation.
08	xxxx080D	CSVDYNEXRSNBADPARMLISTALET **Meaning**: Program error. The system found an error in the ALET for the parameter list mapped by the CSVEXAA macro. **Action**: Ensure that the ALET is 0 or that the ALET represents a valid entry on the DU-AL.
08	xxxx080E	CSVDYNEXRSNBADVERSION **Meaning**: Program error. The system found an incorrect version number in the parameter list that the CSVDYNEX 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 you specified the COMPLETE option on at least one invocation.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx080F	CSVDYNEXRSNLOCKED **Meaning**: Program error. For DEFINE, ADD, MODIFY, REPLACE, DELETE, UNDEFINE, ATTRIB and LIST requests: the caller holds a lock. **Action**: Before you issue CSVDYNEX with the specified request, release any locks held.
08	xxxx0814	CSVDYNEXRSNNOFASTPATH **Meaning**: Program error. The CALL request with FASTPATH=YES is not valid for this exit. **Action**: Redefine the exit to allow FASTPATH processing, or specify FASTPATH=NO on the CALL request.
08	xxxx0815	CSVDYNEXRSNBADDSNAREA **Meaning**: Program error. The system cannot access the data set name. **Action**: Make sure you specify the DSNAME parameter with the correct field.
08	xxxx0816	CSVDYNEXRSNBADRETAREA **Meaning**: Program error. The system cannot access the return area. **Action**: Make sure you specify the RETAREA parameter with the correct field.
08	xxxx0817	CSVDYNEXRSNBADWORKAREA **Meaning**: Program error. The system cannot access the work area. **Action**: Make sure you specify the WORKAREA parameter with the correct field.
08	xxxx0818	CSVDYNEXRSNBADOPEN **Meaning**: Program error. The system is unable to open the specified data set. **Action**: Ensure that: • You specified the DSNAME parameter correctly • The data set is partitioned • The system can locate the data set.
08	xxxx0819	CSVDYNEXRSNEXITNAMENOTFOUND **Meaning**: Program error. For MODIFY, REPLACE, DELETE, CALL, and RECOVER: the system cannot locate the exit name. **Action**: Make sure you specify the correct exit name on the EXITNAME parameter.
08	xxxx081A	CSVDYNEXRSNBADRETLEN **Meaning**: Program error. For a CALL or RECOVER request: the return area is not large enough for even one entry. **Action**: Provide a large enough return area on the RETAREA parameter. The minimum area size can be calculated using the assembler expression L'EXRETHDR+L'EXRETINFO.
08	xxxx081B	CSVDYNEXRSNREG2INRUB **Meaning**: Program error. For a CALL request: an exit routine is called for an exit that is defined AMODE=24 or AMODE=DEFINED. The call specifies a register update block (RUB) that uses register 2. **Action**: Do not specify in the RUB that GPR2 is to be passed to the exit routine.
08	xxxx081C	CSVDYNEXRSNMODULENOTFOUND **Meaning**: Program error. For an ADD, MODIFY, or REPLACE request: the system could not locate the exit routine. **Action**: Make sure you specified the MODNAME parameter correctly.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxyy081D	CSVDYNEXRSNNORESMGR **Meaning**: Program error. The system was unable to establish a resource manager for the exit routine. *yy* contains the return code from RESMGR. **Action**: See the return codes from the RESMGR macro.
08	xxxx081E	CSVDYNEXRSNBADNEXTTOKEN **Meaning**: Program error. For a CALL request: the value you specified on NEXTTOKEN is not valid. **Action**: Make sure the field you specified on the NEXTTOKEN parameter is not overlaid.
08	xxxx081F	CSVDYNEXRSNWORKAREABADDATA **Meaning**: Program error. For a RECOVER request: the work area contains bad data. **Action**: Make sure that: • The field you specified on the WORKAREA parameter field is not overlaid • The work area parameter field is the same one that was specified on the CALL request • The work area parameter field was not changed between the CALL request and the RECOVER request • You zeroed the first word of the work area before issuing the CALL request.
08	xxxx0820	CSVDYNEXRSNBADDSNAMEALET **Meaning**: Program error. The system found an error in the ALET that qualifies the data set name area you specified on the DSNAME parameter. **Action**: Make sure you specified the ALET correctly. If you specified DSNAME=(n), you might not have set up ARn correctly.
08	xxxx0821	CSVDYNEXRSNBADRETAREAALET **Meaning**: Program error. For a CALL request: the system found an error in the ALET that qualifies the return area you specified on the RETAREA parameter. **Action**: Make sure you specified the ALET correctly. If you specified RETAREA=(n), you might not have set up ARn correctly.
08	xxxx0822	CSVDYNEXRSNBADEXITNAME **Meaning**: Program error. For a DEFINE, ADD, MODIFY, REPLACE, DELETE, UNDEFINE, ATTRIB, CALL or RECOVER request: you specified an incorrect exit name on the EXITNAME parameter. The first character is either **0** or blank. **Action**: Correct the exit name.
08	xxxx0823	CSVDYNEXRSNBADMODNAME **Meaning**: Program error. For an ADD, MODIFY, or REPLACE request: you specified an incorrect exit routine name on the MODNAME parameter. The first character is either **0** or blank. **Action**: Correct the exit name.
08	xxxx0824	CSVDYNEXRSNBADRUB **Meaning**: Program error. For a CALL request: the system encountered an error while accessing the RUB. **Action**: Make sure the RUB area is valid.
08	xxxx0825	CSVDYNEXRSNBADRUBALET **Meaning**: Program error. For a CALL request: the system found an error in the ALET that qualifies the RUB area you specified on the RUB parameter. **Action**: Make sure you specified the ALET correctly. If you specified RETAREA=(n), you might not have set up ARn correctly.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx0826	CSVDYNEXRSNBADSDWA **Meaning**: Program error. For a RECOVER request: the system encountered an error while accessing the SDWA passed as a parameter on the RECOVER request. **Action**: Make sure the SDWA address you provided on the SDWA parameter is the one the system provided to the recovery routine.
08	xxxx0827	CSVDYNEXRSNBADAMODE **Meaning**: Program error. For an ADD, MODIFY, or REPLACE request: one of the following occurred: • An exit routine with AMODE=31 is being added to an exit that requires that its exit routines have AMODE=24. • An exit routine with AMODE=24 is being added to an exit that requires that its exit routines have AMODE=31. **Action**: Make sure the AMODE attributes of the exit routine to be added conform to the exit definition.
08	xxxx0828	CSVDYNEXRSNBADKEY **Meaning**: Program error. One of the following: • For DEFINE: the input key you specified on the KEY parameter is not valid. • For CALL FASTPATH=YES: the caller's key does not match the key that the exit requires, according to its definition. **Action**: One of the following: • For DEFINE: specify a valid key. • For CALL FASTPATH=YES: change your key using the MODESET macro to match the key that the exit requires, according to its definition. Or redefine the key requirement for the exit.
08	xxxx0829	CSVDYNEXRSNBADALLOC **Meaning**: Program error. For an ADD, MODIFY, or REPLACE request: the system is unable to allocate the data set you specified on the DSNAME parameter. **Action**: Ensure that: • You specified the proper data set • The data set is partitioned • The data set can be located by the system.
08	xxxx082A	CSVDYNEXRSNNOTREENTRANT **Meaning**: Program error. For an ADD, MODIFY, or REPLACE request: an exit routine that is not reentrant is being added to an exit that requires that its exit routines be reentrant. **Action**: Do not add a non-reentrant exit routine to an exit that is defined to call only reentrant routines.
08	xxxx082C	CSVDYNEXRSNBADABENDCONSEC **Meaning**: Program error. For a DEFINE request, an exit that is defined as FASTPATH=YES and ABENDCONSEC=YES does not accept a PSW key value that is 8 or higher, or ANYKEY=YES. **Action**: Correct your REQUEST=DEFINE request.
08	xxxx082D	CSVDYNEXRSNBADESPIE **Meaning**: Program error. A problem state caller issuing the CALL request with FASTPATH=NO cannot have SPIE or ESPIE routines in effect. **Action**: Do not issue the CALL request when you have SPIE or ESPIE routines in effect.

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

Return Code	Reason Code	Equate Symbol Meaning and Action
08	xxxx082E	CSVDYNEXRSNNOTAPFAUTHORIZED **Meaning**: Program error. For an ADD, MODIFY, or REPLACE request, the system cannot load the exit routine from the data set you specified on the DSNAME parameter. The data set is not APF-authorized. **Action**: Specify a data set that is APF-authorized.
08	xxxx0830	CSVDYNEXRSNBADEXAAVER **Meaning**: For the LIST request, an incorrect EXAAVER value was provided to the system. **Action**: Check for possible storage overlay.
08	xxxx0831	CSVDYNEXRSNANYKEYNOTRENT **Meaning**: For the DEFINE request, ANYKEY=YES was specified without REENTRANT=REQ specified. **Action**: Correct your REQUEST=DEFINE request.
08	xxxx0832	CSVDYNEXRSNBADPOS **Meaning**: For the ADD request, an incorrect POS value was provided to the system. **Action**: Check for possible storage overlay.
08	xxxx0833	CSVDYNEXRSNBADEXRETVER **Meaning**: For the CALL or RECOVER request, an incorrect EXRETVER value was provided to the system. **Action**: Check for possible storage overlay.
08	xxxx0836	CsvdynexRsnDeleteWithoutForce **Meaning**: A DELETE request that did not specify FORCE=YES was made for an exit routine that was added with DELETEFORCE=YES. The request is rejected. **Action**: If you are sure that the exit routine can be deleted from the exit, request the DELETE function again, specifying FORCE=YES
08	xxxx083B	CsvdynexRsnModuleAbove2G **Meaning**: Exit routine is RMODE 64. Exit requires RMODE 24 or 31. **Action**: Avoid trying to add to an exit an exit routine which resides above 2G.
0C	xxxx0C02	CSVDYNEXRSNNOSTORAGE **Meaning**: Environmental error. The system does not have the storage to complete the request. **Action**: Contact the system programmer. There is a common storage shortage.
10	xxxx1001	CSVDYNEXRSNCOMPERROR **Meaning**: System error. **Action**: Record the return and reason codes and contact the appropriate IBM support personnel.

Examples of the CSVDYNEX macro

Note: Of the following examples, numbers 1, 2, 8, 9, and 10 are reentrant. The others can be made reentrant using similar constructs.

Example 1

Define an exit named MYEXIT with the following characteristics:

- Exit routines are to get control in AMODE 31
- FASTPATH=YES is allowed on REQUEST=CALL
- All CALL FASTPATH=YES requests will be in PSW key 2 (or key 0, which is allowed regardless of the value specified via KEY).

```
⋮
         CSVDYNEX REQUEST=DEFINE,EXITNAME=LEX,
               AMODE=31,FASTPATH=YES,KEY=2,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,MF=(E,DYNEXL)
*
* Place code to check return/reason codes here
⋮
⋮
* Data Declarations
LEX      DC    CL16'MYEXIT'
LMOD     DC    CL8'MYMOD'
         CSVEXAA               LIST answer area
         CSVEXRET              Return code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
         CSVDYNEX MF=(L,DYNEXL)
⋮
```

Example 2

Associate exit routine named MYMOD with exit MYEXIT. Make the routine inactive. The load module is in data set 'MY.DSN'. When you want the exit routine to get control, make the routine active.

```
         CSVDYNEX REQUEST=ADD,EXITNAME=LEX,
               MODNAME=LMOD,STATE=INACTIVE,DSNAME=LDSN,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,MF=(E,DYNEXL)
*
* Place code to check return/reason codes here
⋮
⋮
* Change MYMOD to be active. Leave its jobname filtering unchanged.   *
         CSVDYNEX REQUEST=MODIFY,EXITNAME=LEX,
               MODNAME=LMOD,STATE=ACTIVE,JOBNAME=LJOB,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,MF=(E,DYNEXL)
*
* Place code to check return/reason codes here
⋮
* Data Declarations
LEX      DC    CL16'MYEXIT'
LMOD     DC    CL8'MYMOD'
LJOB     DC    XL8'0000000000000000' Jobname is to be unchanged
LDSN     DC    CL44'MY.DSN'
         CSVEXAA               LIST answer area
         CSVEXRET              Return code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
         CSVDYNEX MF=(L,DYNEXL)
```

Example 3

Issue the CSVDYNEX CALL request with FASTPATH=NO processing. Ask the system to return information from the exit routine that has the highest return code. Assume that:

- MYEXIT has been defined to be FASTPATH=NO and AMODE=31
- MYMOD has been associated with exit MYEXIT.

```
⋮
         MVC   LRUBBITS,=X'40040000' Indicate regs 1,13 for exit rtn
         LA    1,LPARMLIST      Address of parameter list for routine.
*                               The parameter list must be set up
*                               prior to the CSVDYNEX invocation
         ST    1,LRUBR1         Save in RUB register 1 slot
         LA    1,LSAVEAREA      Address of save area
         ST    1,LRUBR13        Save in RUB register 13 slot
```

```
         XC    LNEXTTOKEN,LNEXTTOKEN Initialize next token
*
         CSVDYNEX REQUEST=CALL,EXITNAME=LEX,
               FASTPATH=NO,NEXTTOKEN=LNEXTTOKEN,
               RUB=LRUB,RETINFO=HIGHEST,
               RETAREA=LRETAREA,RETLEN==AL4(RETALEN),
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
*
         NC    LRSNCODE,=AL4(CSVDYNEXRSNCODEMASK) And off extra bits
         CLC   LRETCODE,=AL4(CSVDYNEXRC_WARN)  Was there an error
         BH    ERROR1          Yes, process error
         BL    OK              RC=0, have information
         CLC   LRSNCODE,=AL4(CSVDYNEXRSNNOMODULES) Any routines?
         BE    DEFAULT         No routines, do default processing
OK       DS    0H              Process return information
         LA    2,LRETAREA      Get address of return area
         USING EXRET,2         Return information header
```

```
*********************************************************************
* Place code to process return information here.  Because the       *
* request was RETINFO=HIGHEST, only one set of information is       *
* returned. You can look at EXRETABEND to tell if the routine abended.*
* If it abended, EXRETABENDCODE and EXRETABENDRSNCODE are set.       *
* If it did not abend, EXRETCODE, EXRETRSN, and EXRETR1 are set.    *
*********************************************************************
         DROP  2               Release using
DEFAULT  DS    0H              Process default
* Place code to process default here
⋮
ERROR1   DS    0H              Process error
```

```
* Place code to process error here
⋮
RETALEN  EQU   L'EXRETHDR+L'EXRETINFO  Size of return area
*                              This is an area big enough for one
*                              entry which is all that is needed since
*                              RETINFO=HIGHEST is specified.
LRETAREA DS    (RETALEN)CL1    Return area
LEX      DC    CL16'MYEXIT'    Name of exit
LNEXTTOKEN DS  D               Next Token
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
LRUB     DS    0XL12           RUB area
LRUBBITS DS    BL.32           Register bits
LRUBR1   DS    A               Register 1 for exit routine
LRUBR13  DS    A               Register 13 for exit routine
LSAVEAREA DS   18F             Standard save area
LPARMLIST DS   A               Parameter list
         CSVEXRET              Return code information
```

Example 4

Issue the CSVDYNEX CALL request with FASTPATH=NO processing. Ask the system to return all information from the exit routines, limited by the space provided in the area specified on the RETAREA keyword.

```
⋮
         MVC   LRUBBITS,=X'40040000' Indicate regs 1,13 for exit rtn
         LA    1,LPARMLIST      Address of parameter list for routine.
*                                 The parameter list must be set up
*                                 prior to the CSVDYNEX invocation
         ST    1,LRUBR1         Save in RUB register 1 slot
         LA    1,LSAVEAREA      Address of save area
         ST    1,LRUBR13        Save in RUB register 13 slot
         XC    LNEXTTOKEN,LNEXTTOKEN Initialize next token
LAB0     DS    0H
*
         CSVDYNEX REQUEST=CALL,EXITNAME=LEX,
               FASTPATH=NO,NEXTTOKEN=LNEXTTOKEN,
               RUB=LRUB,RETINFO=ALL,
               RETAREA=LRETAREA,RETLEN==AL4(RETALEN),
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
*
         NC    LRSNCODE,=AL4(CSVDYNEXRSNCODEMASK) And off extra bits
         CLC   LRETCODE,=AL4(CSVDYNEXRC_WARN)  Was there an error
         BH    ERROR1          Yes, process error
```

```
         BL    OK              RC=0, have information
         CLC   LRSNCODE,=AL4(CSVDYNEXRSNNOMODULES) Any routines?
         BE    DEFAULT         No routines, do default processing
OK       DS    0H              Process return information
         LA    2,LRETAREA      Get address of return area
         USING EXRET,2         Return information header
         L     3,EXRET#RET     Number of entries in area
```

```
*********************************************************************
* There will be 1 or 2 entries in the return information area due     *
* to the request for RETINFO=ALL (if there were 0, the reason code    *
* of CSVDYNEXRSNNOMODULES would have been returned, and that was      *
* processed earlier). If you were dynamically allocating the return   *
* information area, you could use field EXRET#REM to indicate how     *
* many more entries remain so that you could allocate an area large   *
* enough so that all the remaining exit routines would be called on   *
* the next REQUEST=CALL.                                              *
*********************************************************************
```

```
⋮
PROCENT  DS    0H              Process an entry
*********************************************************************
* Place code to process a return information entry here.             *
* You can look at EXRETABEND to tell if the routine abended.         *
*   If it abended, EXRETABENDCODE and EXRETABENDRSNCODE are set.     *
*   If it did not abend, EXRETCODE, EXRETRSN, and EXRETR1 are set.   *
*********************************************************************
         LA    2,L'EXRETINFO(2) Move to next entry. Note that once
                   this is done, you can no longer reference the
                   fields in area EXRETHDR.
         DROP  2               Release using
         BCT   3,PROCENT       If more entries, continue
         CLC   LRETCODE,=AL4(CSVDYNEXRC_WARN)
         BNE   LAB2            No more exits, done
         CLC   LRSNCODE,=AL4(CSVDYNEXRSNMOREMODULES)
         BE    LAB0            More exits, continue
LAB2     DS    0H              No more exits
* Place code to process no-more-exits here
⋮
DEFAULT  DS    0H              Process default
* Place code to process default here
⋮
ERROR1   DS    0H              Error
* Place code to process errors here
⋮
* Data Declarations
```

```
RETALEN  EQU   L'EXRETHDR+2*L'EXRETINFO  Size of return area
*                                 Room for 2 routines' information
*                                 is provided
LRETAREA DS    (RETALEN)CL1    Return area
LEX      DC    CL16'MYEXIT'    Name of exit
LNEXTTOKEN DS  F               Next Token
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
LRUB     DS    0XL12           RUB area
LRUBBITS DS    BL.32           Register bits
LRUBR1   DS    A               Register 1 for exit routine
LRUBR13  DS    A               Register 13 for exit routine
LSAVEAREA DS   18F             Standard save area
LPARMLIST DS   A               Parameter list
         CSVEXRET              Return code information
⋮
```

Example 5

Issue the CSVDYNEX CALL request with FASTPATH processing. Ask the system to return all information from the exit routines, limited by the space provided in the area specified on the RETAREA keyword. Provide the CSVDYNEX RECOVER request within the recovery code you provide for an abnormally ending exit routine.

```
⋮
* Establish recovery for the exit routine                           *
         ST   12,MYBASEREG       Save basereg for ESTAEX routine
         XC    FOOTPRINT,FOOTPRINT Clear footprints
```

```
         ESTAEX MYESTAEX,CT,PARAM=MYPARAM
⋮
```

```
*                                                                     *
* Set up for fast-path call. Note that it is necessary to             *
* clear the NextToken area (LNEXTTOKEN) prior to the first            *
* REQUEST=CALL and it is necessary to clear the first                 *
* four bytes of the workarea prior to each REQUEST=CALL.              *
*                                                                     *
         MVC   LRUBBITS,=X'40040000' Indicate regs 1,13 for exit rtn
         LA    1,LPARMLIST      Address of parameter list for routine.
*                                 The parameter list must be set up
*                                 prior to the CSVDYNEX invocation
         ST    1,LRUBR1         Save in RUB register 1 slot
         LA    1,LSAVEAREA      Address of save area
         ST    1,LRUBR13        Save in RUB register 13 slot
         XC    LNEXTTOKEN,LNEXTTOKEN Clear token
CALLEXIT DS    0H               Retry label from recovery
         XC    LWORKAREA(4),LWORKAREA  Clear first 4 bytes
```

```
* Issue REQUEST=CALL, specifying FASTPATH processing, and            *
* RETINFO=ALL, meaning information about all exit routines called    *
* will be returned to the RETAREA.                                   *
         LA    2,LRETAREA       Return area
         USING EXRET,2
         XC    EXRET#RET,EXRET#RET  Clear the field. This ensures that
*               if recovery is entered, the return area can be
*               examined. See comment in REQUEST=RECOVER processing.
         DROP  2
         OI    FOOTPRINT,INCSVDYNEX    Set footprint for recovery
         CSVDYNEX REQUEST=CALL,EXITNAME=LEX,
               FASTPATH=YES,NEXTTOKEN=LNEXTTOKEN,
               WORKAREA=LWORKAREA,RUB=LRUB,RETINFO=ALL,
               RETAREA=LRETAREA,RETLEN==AL4(RETALEN),
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
```

```
         NI    FOOTPRINT,X'FF'-INCSVDYNEX Reset footprint for recovery
         NC    LRSNCODE,=AL4(CSVDYNEXRSNCODEMASK) And off extra bits
         CLC   LRETCODE,=AL4(CSVDYNEXRC_WARN)  Was there an error
         BH    ERROR1           Yes, process error
         BL    OK               RC=0, have information
         CLC   LRSNCODE,=AL4(CSVDYNEXRSNNOMODULES) Any routines?
         BE    DEFAULT          No routines, do default processing
OK       DS    0H               Process return information
* Place code to process return information here
⋮
         CLC   LRETCODE,=AL4(CSVDYNEXRC_WARN) Check return code
         BL    LAB2             Must be RC=0, no more exits, done
         CLC   LRSNCODE,=AL4(CSVDYNEXRSNMOREMODULES) Check reason code
         BE    CALLEXIT         More exits, continue
LAB2     DS    0H               No more exits
```

```
* Place code to process no-more-exits here
⋮
         B     ENDCALL          Join common path
DEFAULT  DS    0H               Default
* Place code to process default here
⋮
         B     ENDCALL          Join common path
ERROR1   DS    0H               Error
* Place code to process error here
⋮
         B     ENDCALL          Join common path
ENDCALL  DS    0H               Common path
```

```
         ESTAEX 0               Remove recovery routine
⋮
```

```
         DS    0D               Doubleword align
LWORKAREA DS   CL512            Work area
RETALEN  EQU   L'EXRETHDR+2*L'EXRETINFO  Size of return area.  Room for
*                                 2 routines' information is provided.
LRETAREA DS    (RETALEN)CL1     Return area
LEX      DC    CL16'MYEXIT'     Name of exit
LNEXTTOKEN DS  D                Next Token
LRETCODE DS    F                Return code
```

```
LRSNCODE DS    F               Reason code
LRUB     DS    0XL12           RUB area
LRUBBITS DS    BL.32           Register bits
LRUBR1   DS    A               Register 1 for exit routine
LRUBR13  DS    A               Register 13 for exit routine
LSAVEAREA DS   18F             Standard save area
LPARMLIST DS   A               Parameter list
MYPARAM  DS    0F              ESTAEX parameter area
MYBASEREG DS   F               Base register
FOOTPRINT DS   X               Footprint byte
INCSVDYNEX EQU X'80'           Bit 0 of footprint
⋮
* ESTAEX routine                                                *
```

```
MYESTAEX DS    0H
* ESTAEX routine entry linkage                                  *
         PUSH  USING
         DROP  ,                 Avoid using mainline's regs yet
         USING *,15              Temporary addressability
         LA    3,12              No-SDWA constant
         CR    0,3               Is SDWA provided
         BE    RLAB1             No, branch
         LR    3,1               Save address of SDWA
         L     2,0(1)            Get address of user parameter
         L     2,0(2)            Get address of MYPARAM
         B     RLAB2             Skip No-SDWA path
RLAB1    DS    0H                No SDWA
         SLR   3,3               Set 0 for SDWA address
*                                Reg 2 has address of user parameter
RLAB2    DS    0H
```

```
* ESTABLISH addressability to mainline information              *
         L     12,MYBASEREG-MYPARAM(2) Get basereg
         POP   USING
         ST    14,SAVER14        Save return address
         TM    FOOTPRINT,INCSVDYNEX  Were we within CSVDYNEX?
         BZ    RLAB3             No, skip this
         NI    FOOTPRINT,X'FF'-INCSVDYNEX Turn off footprint
```

```
*********************************************************************
* Issue REQUEST=RECOVER. Note that this simplified example does not  *
* examine return information from CSVDYNEX CALL to determine where   *
* the error occurred.                                                *
*                                                                    *
* The NEXTTOKEN returned should be used on the next REQUEST=CALL     *
*   when the return and reason codes indicated that there were       *
*   more routines to call.                                           *
* The return information (RETAREA) does not indicate (in field       *
*   EXRET#REM) how many more routines remain to be called.           *
* The RETAREA field passed on REQUEST=CALL might have information in *
* it as well. If you zeroed the EXRET#RET field prior to the call,the*
* return area can be examined (except for EXRET#REM) as can be done  *
* for REQUEST=CALL - i.e., using the EXRET#RET value to indicate     *
* the number of return information entries that are present          *
* (when REQUEST=CALL specified RETINFO=ALL) or the number of         *
* exit routines called (when REQUEST=CALL specified HIGHEST, LOWEST  *
*                                                                    *
* Information is returned for each routine called, limited by amount *
*   of space provided in the area specified on the RETAREA keyword.  *
*********************************************************************
         CSVDYNEX REQUEST=RECOVER,EXITNAME=LEX,
               NEXTTOKEN=LNEXTTOKEN,WORKAREA=LWORKAREA,
               RETAREA=RRETAREA,RETLEN==AL4(RRETALEN),SDWA=0(3),
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
```

```
         NC    LRSNCODE,=AL4(CSVDYNEXRSNCODEMASK) Clear unused bits
         CLC   LRETCODE,=AL4(CSVDYNEXRC_WARN)
         BH    RERROR1           RC>4 => error, stop calling
         BL    RLAB3             RC=0 => no more routines, CALL is done
         CLC   LRSNCODE,=AL4(CSVDYNEXRSNMOREMODULES) Check reason code
         BNE   RLAB3             No more routines, done with REQUEST=CALL
         LA    15,CALLEXIT       Retry label
         ST    15,RETADDR        Save it
         B     PROCREC           Join common recovery
RLAB3    DS    0H                No REQUEST=RECOVER needed
         LA    15,ENDCALL        Retry label
         ST    15,RETADDR        Save it
PROCREC  DS    0H                Process REQUEST=RECOVER return info
```

```
         LA    2,RRETAREA      Get address of return area
         USING EXRET,2         Return information header

*********************************************************************
* Place code to process return information here.                    *
* There will be only one set of information returned.               *
* You can look at EXRETRECOVERFLAGS to tell what happened.          *
* Those flags can indicate whether there was a problem with the     *
* exit routine or with the interface information that you provided.  *
*********************************************************************
         DROP  2               Release using
         B     COMMON          Join common code
RERROR1  DS    0H              Error case
* Put code here to process error case                              *
⋮
         LA    15,ENDCALL      Retry label
         ST    15,RETADDR      Save it
COMMON   DS    0H              No REQUEST=RECOVER needed

* Put code here to process the rest of the recovery                *
⋮

EXIT     DS    0H              Exit from recovery
         L     4,RETADDR       Get retry address
         LTR   3,3             See if SDWA exists
         BZ    RLAB6           No SDWA, branch
         SETRP WKAREA=(3),RC=4,RETADDR=(4),FRESDWA=YES
         B     RLAB7
RLAB6    DS    0H              No SDWA
         LR    0,4             Set retry address
         LA    15,4            Indicate to retry
RLAB7    DS    0H              Exit from recovery
         L     14,SAVER14      Restore return address
         BR    14              Return
SAVER14  DS    A               Saved return address
RETADDR  DS    A               Retry address
RRETALEN EQU   L'EXRETHDR+L'EXRETINFO Size of return area
RRETAREA DS    (RETALEN)CL1    Return area
         DS    0D
         CSVEXAA               LIST answer area
         CSVEXRET              Return code information
         IHASDWA
```

Example 6

List information about all exits in the system.

```
⋮
         L     2,=AL4(INITEXAA)  Initial answer area size
         ST    2,SIZEEXAA      Save it
         GETMAIN RU,LV=(2)     Allocate the answer area
         ST    1,EXAA@         Save address of answer area
LAB1     DS    0H
         L     4,EXAA@         Address of answer area

*********************************************************************
*        Issue the CSVDYNEX LIST request                            *
*********************************************************************
         CSVDYNEX REQUEST=LIST,ANSAREA=(4),ANSLEN=SIZEEXAA,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
*
         CLC   LRETCODE(4),=AL4(CSVDYNEXRC_WARN) Warning?
         BNE   LAB2            No, request successful or error
*                              Yes, not enough room
         LR    3,2             Save current size
         L     2,EXAAHTLEN-EXAAHDR(4) Get required size
         FREEMAIN RU,A=(4),LV=(3) Release old area
         ST    2,SIZEEXAA      Save it
         GETMAIN RU,LV=(2)     Allocate new area
         ST    1,EXAA@         Save address of answer area
         B     LAB1            Retry List operation
LAB2     DS    0H
         CLC   LRETCODE(4),=AL4(CSVDYNEXRC_OK) Success?
         BNE   LAB3            No, error
```

```
* Process information in answer area when RC=0                        *
         USING EXAAHDR,4       EXAAHDR DSECT
         L     5,EXAAH#REC     Find how many EXAAE entries
         LTR   5,5             Are there any entries
         BZ    LAB4            No, join common path
         L     4,EXAAHFIRST@   Get first entry
         USING EXAAE,4         EXAAE DSECT
LAB5     DS    0H              EXAAE loop
*
* Put code to process information contained in EXAAE here
⋮
         LH    7,EXAAE#ENT     Get number of EXAAM entries
         N     7,CLEAR0TO15    Clear bits 0 to 15
*                              Are there any routines
         BZ    LAB7            No, move to end of EXAAE loop
         L     6,EXAAEFIRSTENT@ Get first EXAAM
LAB6     DS    0H              EXAAM loop
         USING EXAAM,6         EXAAM DSECT
```

```
* Put code to process information contained in EXAAM here
⋮
         L     6,EXAAMNEXT@    Get next EXAAM
         DROP  6               EXAAM DSECT
         BCT   7,LAB6          Continue while there are more
LAB7     DS    0H              Bottom of EXAAE loop
         L     4,EXAAENEXT@    Get next EXAAE
         BCT   5,LAB5          Continue while there are more
         B     LAB4            Skip error case
LAB3     DS    0H              Error return
* Process error case                                                  *
⋮
LAB4     DS    0H              Common path
         L     2,SIZEEXAA      Get size of area
         L     4,EXAA@         Get address of area
         FREEMAIN RU,A=(4),LV=(2) Release area
⋮
```

```
* Data Declarations
EXAA@    DS    A               Address of answer area
SIZEEXAA DS    F               Size of answer area
TEMPSIZE DS    F               Temporary
MODLEN   EQU   20*EXAAM_LEN    Room for 20 routines' information
EXLEN    EQU   10*EXAAE_LEN    Room for 10 exits' information
INITEXAA EQU   EXAAHDR_LEN+MODLEN+EXLEN  Initial size of answer area
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
CLEAR0TO15 DC  X'0000FFFF'     Mask to clear bits 0-15
         CSVEXAA               LIST answer area
         CSVEXRET              Return code information
```

Example 7

Determine if a particular exit has any routines associated with it at this moment. You might use this request if the setup for the CSVDYNEX CALL was very extensive.

```
⋮
         CSVDYNEX REQUEST=QUERY,EXITNAME=LEX,QTYPE=CALL,
               WORKAREA=LWORKAREA,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE
         NC    LRSNCODE,=AL4(CSVDYNEXRSNCODEMASK) Clear unused bits
         CLC   LRETCODE,=AL4(CSVDYNEXRC_WARN)
         BE    LAB1            Warning, didn't find any routines
         BH    ERROR1          Some sort of error
*                              Otherwise, RC=0
```

```
* Place code to set up and call the exit using REQUEST=CALL here
⋮
         B     LAB9            Join common path
LAB1     DS    0H              No routines found
*          Possible reason codes at this point are:
*            CSVEXRETRSNNOMODULES,
*            CSVEXRETRSNQUERYNOTFOUND, and
*            CSVEXRETRSNIMPLICITLYDEFINED.
* Place code to process default here
*
```

```
         B     LAB9            Join common path
ERROR1   DS    0H              Error
*
```

```
* Handle other conditions
⋮
* Data Declarations
LAB9     DS    0H              End of processing
         DS    0D              Doubleword align
LWORKAREA DS   CL512           Work area
LEX      DC    CL16'MYEXIT'    Name of exit
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
         CSVEXAA               LIST answer area
         CSVEXRET              Return code information
```

Example 8

Tell the system how to handle information that returns from multiple routines. In this example, if the return code from an exit routine is greater than the value 4 (KEEPRCCOMP=GT and KEEPRC parameters), and the issuer of CSVDYNEX REQUEST=CALL specified RETINFO=LAST, RETINFO=LOWEST, or RETINFO=HIGHEST, the system keeps the return code and passes it back to the caller of the exit.

```
⋮
         CSVDYNEX REQUEST=ATTRIB,EXITNAME=LEX,
               KEEPRC=KEEPRCCVAL,KEEPRCCOMP=GT,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,MF=(E,MYPLIST)
* CHECK return codes
⋮
```

```
* Data Declarations
LEX        DC    CL16'MYEXIT'    Name of exit
KEEPRCCVAL DC   F'4'              Keep any return code > this
DYNAREA    DSECT
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
         CSVDYNEX MF=(L,MYPLIST) Define storage
         CSVEXAA               LIST answer area
         CSVEXRET              Return code information
```

Example 9

Delete routine MYMOD from exit MYEXIT.

```
⋮
         CSVDYNEX REQUEST=DELETE,EXITNAME=LEX,
               MODNAME=LMOD,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,MF=(E,DYNEXL)
*
* Place code to check return/reason codes here
⋮
* Data Declarations
LEX      DC    CL16'MYEXIT'
LMOD     DC    CL8'MYMOD'
         CSVEXAA               LIST answer area
         CSVEXRET              Return code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
         CSVDYNEX MF=(L,DYNEXL)
```

Example 10

Remove the definition of MYEXIT.

```
         CSVDYNEX REQUEST=UNDEFINE,EXITNAME=LEX,
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,MF=(E,DYNEXL)
* Place code to check return/reason codes here
⋮
* Data Declarations
LEX      DC    CL16'MYEXIT'
```

```
         CSVEXAA              LIST answer area
         CSVEXRET             Return code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
         CSVDYNEX MF=(L,DYNEXL)
```

CSVDYNEX - List form

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

Syntax	Description
name	*name*: symbol. Begin *name* in column 1.
␣	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
␣	One or more blanks must follow CSVDYNEX.
,PLISTVER=IMPLIED_VERSION	
,PLISTVER=MAX	**Default**: IMPLIED_VERSION
,PLISTVER=*plistver*	*plistver*: 0 - 1
MF=(L,*list addr*)	*list addr*: symbol.
MF=(L,*list addr,attr*)	*attr*: 1- to 60-character input string.
MF=(L,*list addr*,0D)	**Default:** 0D

Parameters

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

CSVDYNEX - Modify form

Use the modify form of the CSVDYNEX 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.

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNEX.
CSVDYNEX	
ƀ	One or more blanks must follow CSVDYNEX.
	The parameters on the modify form are identical to those on the standard form with the following exceptions:
	- FASTPATH is not valid on the modify form.
	- MF, COMPLETE, and NOCHECK are listed below.
	The following list tells you where to look for the syntax diagrams of the standard form of CSVDYNEX.
	REQUEST=DEFINE — in “Define an exit” on page 359
	REQUEST=ADD — in “Add an exit routine to an exit” on page 366
	REQUEST=MODIFY — in “Change the state of an exit routine” on page 373
	REQUEST=DELETE — in “Delete an exit routine from an exit” on page 376
	REQUEST=UNDEFINE — in “Remove the definition of an exit” on page 379
	REQUEST=ATTRIB — in “Change the attributes of an exit” on page 382
	REQUEST=LIST — in “List information about one or more exits” on page 385
	REQUEST=CALL — in “Call one or more exit routines at an exit” on page 389
	REQUEST=RECOVER — in “Provide recovery for an exit routine that abnormally ended” on page 395

Syntax	Description
	REQUEST=QUERY — in “Determine whether an exit routine exists for an exit” on page 399
,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 CSVDYNEX are described in the standard form of the macro with the following exceptions:

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

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

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

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

When using the NOCHECK option with the modify or execute forms, be sure that it is preceded by a modify or execute form invocation that specifies or defaults to the COMPLETE option. This ensures that the parameter list is initialized completely.

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

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

CSVDYNEX - Execute form

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

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

Syntax	Description
ƀ	One or more blanks must follow CSVDYNEX.
	The parameters on the execute form are identical to those on the standard form with the exception of the MF, COMPLETE, and NOCHECK parameters listed below.
	The following list tells you where to look for the syntax diagrams of the standard form of CSVDYNEX.
	REQUEST=DEFINE — in “Define an exit” on page 359
	REQUEST=ADD — in “Add an exit routine to an exit” on page 366
	REQUEST=MODIFY — in “Change the state of an exit routine” on page 373
	REQUEST=DELETE — in “Delete an exit routine from an exit” on page 376
	REQUEST=UNDEFINE — in “Remove the definition of an exit” on page 379
	REQUEST=ATTRIB — in “Change the attributes of an exit” on page 382
	REQUEST=LIST — in “List information about one or more exits” on page 385
	REQUEST=CALL — in “Call one or more exit routines at an exit” on page 389
	REQUEST=RECOVER — in “Provide recovery for an exit routine that abnormally ended” on page 395
	REQUEST=QUERY — in “Determine whether an exit routine exists for an exit” on page 399
	The REQUEST parameter is required, even when you specify the NOCHECK parameter.
,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 CSVDYNEX macro with the following exceptions:

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

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

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

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

Chapter 39. CSVDYNL — Provide dynamic LNKLST services

CSVDYNL provides an interface to request dynamic LNKLST services. With CSVDYNL, you can request services for the following operations:

- Define a LNKLST set that can be used as the LNKLST concatenation (REQUEST=DEFINE).
- Add a data set to a LNKLST set (REQUEST=ADD).
- Delete a data set from a LNKLST set (REQUEST=DELETE).
- Remove the definition of a LNKLST set (REQUEST=UNDEFINE).
- Test to determine if a module can be located in a LNKLST set (REQUEST=TEST).
- Obtain a list of LNKLST sets and users (REQUEST=LIST).
- Update jobs and address spaces to use the current LNKLST set (REQUEST=UPDATE).
- Query information about support for LNKLST services (REQUEST=QUERYDYN).

Following the descriptions of the standard forms of all requests are:

- The return and reason codes, see “Return and reason codes” on page 468.
- Examples of using CSVDYNL, see “Examples” on page 474.

REQUEST=DEFINE option of CSVDYNL

REQUEST=DEFINE allows you to define a LNKLST set for the LNKLST concatenation.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	SAF authority to the FACILITY class. If SAF is not available, or has no pertinent information, then the caller must be supervisor state or PKM 0-7, or PSW key 0-7, or APF-authorized. The SAF entity name and authorization level applied is UPDATE authority to FACILITY class entity CSVDYNL.lnklstname.DEFINE
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
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

The caller should include the CSVDLAA macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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 if GPR15 is not 0

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 CSVDYNL macro is written as follows:

Syntax	Description

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNL.
CSVDYNL	
ƀ	One or more blanks must follow CSVDYNL.
REQUEST=DEFINE	
,LNKLSTNAME=*lnklstname*	*lnklstname:* RS-type address or address in register (2) - (12).
,COPYFROM=*copyfrom*	*copyfrom:* RS-type address or address in register (2) - (12).
,COPYFROM=NO_COPY	**Default:** COPYFROM=NO_COPY
,CHECKSYS1=YES	**Default:** CHECKSYS1=YES
,CHECKSYS1=NO	
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,MF=(E,*list addr*,NOCHECK)	
,MF=(M,*list addr*)	
,MF=(M,*list addr*,COMPLETE)	

Syntax	Description
,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 CSVDYNL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=DEFINE
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=DEFINE indicates to define a LNKLST set.

,LNKLSTNAME=*lnklstname*
A required input parameter that contains the name of the LNKLST set. For IBM-provided LNKLST sets, this name should begin with the letters SYS. The first character must not be X'00' or blank. It is recommended that the name use characters from among the set of alphanumerics, special (@#$), underscore, and period. There should be no imbedded blanks. Avoid using the names "CURRENT" and "IPL". The first is reserved to mean "the current LNKLST". The other is reserved to mean the LNKLST defined via the LNKLSTxx parmlib members and the LNK parameter of the IEASYSxx parmlib member.

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

,COPYFROM=*copyfrom*
,COPYFROM=NO_COPY
An optional input parameter that contains the name of the LNKLST set to be copied in order to initialize the LNKLST set being defined. If "CURRENT" is specified, the current LNKLST set will be used. By default, the LNKLST set being defined will contain the LINKLIB, MIGLIB, CSSLIB, LINKLIBE and MIGLIBE data sets (SYS1.LINKLIB, SYS1.MIGLIB, SYS1.CSSLIB, SYS1.SIEALNKE, and SYS1.SIEAMIGE, unless overridden by the SYSLIB statement of PROGxx). The default is NO_COPY.

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

,CHECKSYS1=YES
,CHECKSYS1=NO
An optional parameter that indicates whether the LNKLST set must contain SYS1.LINKLIB, SYS1.MIGLIB, SYS1.CSSLIB, SYS1.SIEALNKE, and SYS1.SIEAMIGE. The default is CHECKSYS1=YES.

,CHECKSYS1=YES
specifies that the LNKLST set must contain SYS1.LINKLIB, SYS1.MIGLIB, and SYS1.CSSLIB.

,CHECKSYS1=NO
specifies that the LNKLST set does not need to contain one or more of the following:

- SYS1.LINKLIB
- SYS1.MIGLIB
- SYS1.CSSLIB

When CHECKSYS1=NO is specified, it is the customer's responsibility to have an alternate data set for each SYS1.xxxLIB data set that is not in the LNKLST set, and that data set must contain all of the information present in the SYS1.xxxLIB data set. Be aware that all LNKLST sets begin with the LINKLIB, MIGLIB, and CSSLIB data sets as defined by the SYSLIB statement of the PROGxx parmlib member. These data sets default to SYS1.xxxLIB. Thus the only way to create a LNKLST set without one or more of the SYS1.xxxLIB data sets is to have used the SYSLIB statement to provide a different name for those data sets.

This option should be used with care. It might be used, for instance, to allow for compressing of SYS1.LINKLIB after additional members have been added to it. Since compressing can only safely

be done when the data set is not allocated, the data set must not be in use as an active LNKLST. The following protocol could be used, for instance:

- Use the SYSLIB statement of the PROGxx parmlib member during IPL to define alternate names for the LINKLIB, MIGLIB, and CSSLIB data sets.
- Create a data set that contains a copy of SYS1.LINKLIB.
- DEFINE a LNKLST set that is the same as the current one, but has the new data set in place of SYS1.LINKLIB.
- ACTIVATE that new LNKLST set.
- Stop the library lookaside (LLA) facility.
- UPDATE jobs to use the new LNKLST set. There are cautions associated with using the UPDATE function that you need to be aware of. After doing the UPDATE and stopping LLA, SYS1.LINKLIB should no longer be allocated.
- Compress SYS1.LINKLIB.
- ACTIVATE the previous LNKLST set.
- UPDATE jobs to use that LNKLST set.
- Start LLA.

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

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

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

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,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 CSVDYNL in the following order:

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 468 for the return and reason codes.

Examples

See "Examples" on page 474 for an example.

REQUEST=ADD option of CSVDYNL

REQUEST=ADD allows you to add a data set to a LNKLST set.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	SAF authority to the FACILITY class. If SAF is not available, or has no pertinent information, then the caller must be supervisor state or PKM 0-7, or PSW key 0-7, or APF-authorized. The SAF entity name and authorization level applied is UPDATE authority to FACILITY class entity CSVDYNL.lnklstname.ADD
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided data set name via the DSNAME parameter has the same requirements and restrictions as the control parameters. The user-provided data set name via the AFTERDSNAME parameter has the same requirements and restrictions as the control parameters. The user-provided data set name via the PROBDSNAME parameter has the same requirements and restrictions as the control parameters.

Programming requirements

The caller should include the CSVDLAA macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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 if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CSVDYNL macro is written as follows:

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

Syntax	Description
REQUEST=ADD	
,LNKLSTNAME=*lnklstname*	*lnklstname:* RS-type address or address in register (2) - (12).
,DSNAME=*dsname*	*dsname:* RS-type address or address in register (2) - (12).
,VOLUME=*volume*	*volume:* RS-type address or address in register (2) - (12).
,VOLUME=CATALOG	**Default:** VOLUME=CATALOG
,POS=BOTTOM	**Default:** POS=BOTTOM
,POS=TOP	
,POS=AFTER	
,AFTERDSNAME=*afterdsname*	*afterdsname:* RS-type address or address in register (2) - (12).
,CHECKCONCAT=NO	**Default**: CHECKCONCAT=NO
,CHECKCONCAT=YES	
,PROBDSNAME=*probdsname*	*probdsname:* RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12)
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	

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 CSVDYNL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=ADD
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=ADD indicates to add a data set to a LNKLST set. You cannot add a data set to any active LNKLST set (either the current set, or prior sets that are still in use). You cannot add a data set to LNKLST sets "CURRENT" and "IPL."

Note that if any data set in the LNKLST set is migrated, the issuer's unit of work will wait until the data set is retrieved before continuing.

The system keeps track of the volume on which the data set resides as well as whether or not the data set is managed by the storage management subsystem (SMS). Once the system has determined these values for a data set within a LNKLST set, any of the following will result in an error when the system attempts to allocate the LNKLST set:

- A data set has changed from not SMS-managed to SMS-managed, or vice versa;
- A data set that is not SMS-managed is deleted and then reallocated on another volume

The system flags these cases as errors because they might indicate that the LNKLST set is not what the user expects, and in particular, the APF authorization of the data set might not be as expected. In both cases, if you do want the new data set, you must delete the data set from the LNKLST set and then re-add it.

,LNKLSTNAME=*lnklstname*
A required input parameter that contains the name of the LNKLST set.

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

,DSNAME=*dsname*
A required input parameter that contains the name of the data set or library to be added to the LNKLST set. The data set must be cataloged. It may be allocated as a PDS or as a PDSE.

Note that if the data set is migrated, the issuer's unit of work will wait until the data set is retrieved before continuing.

To code: Specify the RS-type address, or address in register (2) - (12), of a 44-character field.

,VOLUME=*volume*
,VOLUME=CATALOG
An optional input parameter that contains the name of the volume on which the data set resides. Be aware that even when this option is specified, the data set must reside on the volume indicated by the "normal" catalog. If used during IPL when only the master catalog is available, the volume ID will be checked once the "normal" catalog is available. The ADD will be rejected if the specified volume did not match the volume indicated by the catalog. You can use a value of "******" to indicate that the data set is located on the current SYSRES volume. You can use a value of "*MCAT*" to indicate that the data set is located on the volume containing the master catalog. The default is CATALOG.

To code: Specify the RS-type address, or address in register (2) - (12), of a 6-character field.

,POS=BOTTOM
,POS=TOP
,POS=AFTER
An optional parameter that indicates where in the list to place the data set. The default is POS=BOTTOM.

,POS=BOTTOM
specifies to place the data set at the bottom or end of the list.

,POS=TOP
specifies to place the data set at the top or start of the list. Note that the system always places the LINKLIB, MIGLIB, CSSLIB, SIEALNKE, and SIEAMIGE data sets at the top of the list. Thus POS=TOP would indicate to place the data set immediately after the CSSLIB data set.

,POS=AFTER
specifies to place the data set after the data set named by the AFTERDSNAME parameter. Note that you cannot place the data set in between two of the system libraries that are always present in the LNKLST. Thus you cannot specify the LINKLIB or MIGLIB data set via the AFTERDSNAME parameter. You also cannot specify the CSSLIB data set via "AFTER=xx". Use POS=TOP if you want the data set to be placed into the LNKLST immediately after the CSSLIB data set.

,AFTERDSNAME=*afterdsname*
When POS=AFTER is specified, a required input parameter that contains the name of the data set or library after which the data set named by the DSNAME parameter is to be placed within the LNKLST set. The data set must be cataloged.

To code: Specify the RS-type address, or address in register (2) - (12), of a 44-character field.

,CHECKCONCAT=NO
,CHECKCONCAT=YES
An optional parameter that indicates whether to check if the concatenation defined by the LNKLST set is full. The default is CHECKCONCAT=NO.

,CHECKCONCAT=NO
specifies not to check if the concatenation is full. If the concatenation actually is full, the situation will be caught when the LNKLST set is activated.

,CHECKCONCAT=YES
specifies to check if the concatenation is full. This implies that all the data sets in the LNKLST set must be allocated and concatenated together. The system processing for this option will be longer than when CHECKCONCAT=NO is specified.

,PROBDSNAME=*probdsname*
An optional output parameter that is to contain the name of the "problem" data set or library for which processing failed. The library either could not be allocated, opened, or caused the extent limit to be exceeded.

To code: Specify the RS-type address, or address in register (2) - (12), of a 44-character field.

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=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 CSVDYNL in the following order:

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See "Return and reason codes" on page 468 for the return and reason codes.

Examples

See "Examples" on page 474 for an example.

REQUEST=DELETE option of CSVDYNL

REQUEST=DELETE deletes a data set from a LNKLST set.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	SAF authority to the FACILITY class. If SAF is not available, or has no pertinent information, then the caller must be supervisor state or PKM 0-7, or PSW key 0-7, or APF-authorized. The SAF entity name and authorization level applied is UPDATE authority to FACILITY class entity CSVDYNL.lnklstname.DELETE
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided data set name via the DSNAME parameter has the same requirements and restrictions as the control parameters.

Programming requirements

The caller should include the CSVDLAA macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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 if GPR15 is not 0

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

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

Performance implications

None.

Syntax

The CSVDYNL macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNL.
CSVDYNL	
ƀ	One or more blanks must follow CSVDYNL.
REQUEST=DELETE	
,LNKLSTNAME=*lnklstname*	*lnklstname:* RS-type address or address in register (2) - (12).
,DSNAME=*dsname*	*dsname:* RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,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 CSVDYNL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=DELETE
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=DELETE indicates to remove a data set from the LNKLST set. You cannot delete a data set from any active LNKLST set (either the current set, or prior sets that are still in use). You cannot delete a data set from LNKLST sets "CURRENT" and "IPL".

,LNKLSTNAME=*lnklstname*
A required input parameter that contains the name of the LNKLST set.

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

,DSNAME=*dsname*
A required input parameter that contains the name of the data set or library to be deleted from the LNKLST set.

To code: Specify the RS-type address, or address in register (2) - (12), of a 44-character field.

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=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 CSVDYNL in the following order:

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 468 for the return and reason codes.

Examples

See “Examples” on page 474 for an example.

REQUEST=UNDEFINE option of CSVDYNL

REQUEST=UNDEFINE removes the definition of a LNKLST set. For further information, see the topic “Removing or Compressing a Dataset in an Active LNKLST Set” in *z/OS MVS Initialization and Tuning Reference*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	SAF authority to the FACILITY class. If SAF is not available, or has no pertinent information, then the caller must be supervisor state or PKM 0-7, or PSW key 0-7, or APF-authorized. The SAF entity name and authorization level applied is UPDATE authority to FACILITY class entity CSVDYNL.lnklstname.UNDEFINE
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
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

The caller should include the CSVDLAA macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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 if GPR15 is not 0

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 CSVDYNL macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNL.
CSVDYNL	
ƀ	One or more blanks must follow CSVDYNL.
REQUEST=UNDEFINE	
,LNKLSTNAME=*lnklstname*	*lnklstname:* RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).

Syntax	Description
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,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 CSVDYNL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=UNDEFINE
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=UNDEFINE indicates to remove the definition of a LNKLST set. You cannot remove a LNKLST set that is currently in use. You cannot remove the current LNKLST set. You cannot remove LNKLST set "IPL".

,LNKLSTNAME=*lnklstname*
A required input parameter that contains the name of the LNKLST set.

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

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=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 CSVDYNL in the following order:

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 468 for the return and reason codes.

Examples

See “Examples” on page 474 for an example.

REQUEST=TEST option of CSVDYNL

REQUEST=TEST allows you to determine if a routine can be located using a LNKLST set.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	SAF authority to the FACILITY class. If SAF is not available, or has no pertinent information, then the caller must be supervisor state or PKM 0-7, or PSW key 0-7, or APF-authorized. The SAF entity name and authorization level applied is READ authority to FACILITY class entity CSVDYNL.lnklstname.TEST
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.

Environmental factor	Requirement
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided data set name via the PROBDSNAME parameter has the same requirements and restrictions as the control parameters. The user-provided data set name via the FOUNDDSNAME parameter has the same requirements and restrictions as the control parameters.

Programming requirements

The caller should include the CSVDLAA macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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 if GPR15 is not 0

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 CSVDYNL macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNL.
CSVDYNL	
ƀ	One or more blanks must follow CSVDYNL.
REQUEST=TEST	
,LNKLSTNAME=*lnklstname*	*lnklstname:* RS-type address or address in register (2) - (12).
,MODNAME=*modname*	*modname:* RS-type address or address in register (2) - (12).
,FOUNDDSNAME=*founddsname*	*founddsname:* RS-type address or address in register (2) - (12).
,PROBDSNAME=*probdsname*	*probdsname:* RS-type address or address in register (2) - (12).
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S

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 CSVDYNL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=TEST
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=TEST indicates to test the particular LNKLST set to see if a given routine can be located using it.

Note that if any data set in the LNKLST set is migrated, the issuer's unit of work will wait until the data set is retrieved before continuing.

,LNKLSTNAME=*lnklstname*
A required input parameter that contains the name of the LNKLST set. If "CURRENT" is specified, the current LNKLST set will be used.

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

,MODNAME=*modname*
A required input parameter that contains the name of the routine that is to be located via the LNKLST set. The first character must not be X'00' or blank.

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

,FOUNDDSNAME=*founddsname*
An optional output parameter that is to contain the name of the data set or library in which the requested module was found. This field is only valid when the return code is 0.

To code: Specify the RS-type address, or address in register (2) - (12), of a 44-character field.

,PROBDSNAME=*probdsname*
An optional output parameter that is to contain the name of the "problem" data set or library for which processing failed. The library either could not be allocated, opened, or caused the extent limit to be exceeded.

To code: Specify the RS-type address, or address in register (2) - (12), of a 44-character field.

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=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 CSVDYNL in the following order:

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 468 for the return and reason codes.

Examples

See “Examples” on page 474 for an example.

REQUEST=LIST option of CSVDYNL

REQUEST=LIST returns a list of LNKLST sets and their users.

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:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.

Environmental factor	Requirement
Control parameters:	Control parameters must be in the primary address space or, for AR-mode callers, must be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL). The user-provided answer area (via the ANSAREA parameter) has the same requirements and restrictions as the control parameters.

Programming requirements

The caller should include the CSVDLAA macro to get equate symbols for the return and reason codes.

The caller must include the CSVDLAA macro to get a mapping of the output area provided via the ANSAREA parameter.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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 if GPR15 is not 0

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 CSVDYNL macro is written as follows:

Syntax	Description
name	*name:* symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CSVDYNL.
CSVDYNL	
ƀ	One or more blanks must follow CSVDYNL.
REQUEST=LIST	
,ANSAREA=*ansarea*	*ansarea:* RS-type address or address in register (2) - (12).
,ANSLEN=*anslen*	*anslen:* RS-type address or address in register (2) - (12).
,SEARCH=BYNAME	**Default:** SEARCH=BYNAME
,SEARCH=BYJOBASID	
,LISTLNKNAME=*listlnkname*	*listlnkname:* RS-type address or address in register (2) - (12).
,LISTLNKNAME=ALL_LNKLSTS	**Default:** LISTLNKNAME=ALL_LNKLSTS
,USERINFO=NO	**Default:** USERINFO=NO
,USERINFO=YES	
,JOBNAME=*jobname*	*jobname:* RS-type address or address in register (2) - (12).
,ASID=*asid*	*asid:* 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).

Syntax	Description
,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)	
,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 CSVDYNL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=LIST
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=LIST indicates to list information about the LNKLST sets.

,ANSAREA=*ansarea*
A required output parameter that is to contain the information associated with the LNKLST sets. The area is mapped by macro CSVDLAA.

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

,ANSLEN=*anslen*
A required input parameter that contains the length of the provided answer area.

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

,SEARCH=BYNAME
,SEARCH=BYJOBASID
An optional parameter that indicates what the search criteria are. The default is SEARCH=BYNAME.

,SEARCH=BYNAME
specifies to search for particular LNKLST sets. Information returned is mapped by structure DLAALS (and its subsidiary structures).

,SEARCH=BYJOBASID
specifies to search for particular job or jobs or an ASID. Information returned is mapped by structure DLAAJA. You must specify a valid jobname or ASID.

,LISTLNKNAME=*listlnkname*
,LISTLNKNAME=ALL_LNKLSTS
When SEARCH=BYNAME is specified, an optional input parameter that contains the name of the LNKLST set. If the LNKLST set name is not provided, information about all LNKLST sets is returned. There should be no imbedded blanks. If the LNKLST set name contains wildcard characters (? or *), information is provided about all LNKLST sets that have names that match the provided pattern. If "CURRENT" is specified, the current LNKLST set will be used. Specify "IPL" to get the IPL-time LNKLST if the LNKLST was defined via the LNK parameter of the IEASYSxx parmlib member and the LNKLSTxx parmlib member(s). The default is ALL_LNKLSTS.

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

,USERINFO=NO
,USERINFO=YES
When SEARCH=BYNAME is specified, an optional parameter that indicates whether or not to return information about the users of a LNKLST set. The default is USERINFO=NO.

,USERINFO=NO
specifies not to return user information.

,USERINFO=YES
specifies to return user information.

,JOBNAME=*jobname*
,ASID=*asid*
When SEARCH=BYJOBASID is specified, a required input parameter.

,JOBNAME=*jobname*
A parameter that contains the name of the job that is to be looked for. If the jobname set name contains wildcard characters (? or *), information will be returned about all jobs that match the provided pattern. A jobname in which the first character is blank or hexadecimal zeroes is treated as if JOBNAME had not been specified. The jobname used in performing the comparison is the name of the job for an initiated job, or the name of the address space otherwise.

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

,ASID=*asid*
A parameter that contains the ASID. An ASID of 0 is treated as if ASID had not been specified.

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

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=0
An optional input parameter that specifies the version of the macro. PLISTVER determines which parameter list the system generates. PLISTVER is an optional input parameter on all forms of the macro, including the list form. When using PLISTVER, specify it on all macro forms used for a request and with the same value on all of the macro forms. The values are:

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

  If you can tolerate the size change, IBM recommends that you always specify PLISTVER=MAX on the list form of the macro. Specifying MAX ensures that the list-form parameter list is always long

enough to hold all the parameters you might specify on the execute form, when both are assembled with the same level of the system. In this way, MAX ensures that the parameter list does not overwrite nearby storage.

- **0**, if you use the currently available parameters.

To code: Specify one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0

,MF=S
,MF=(L,*list addr*)
,MF=(L,*list addr,attr*)
,MF=(L,*list addr*,0D)
,MF=(E,*list addr*)
,MF=(E,*list addr*,COMPLETE)
,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 CSVDYNL in the following order:

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 468 for the return and reason codes.

Examples

See “Examples” on page 474 for an example.

REQUEST=UPDATE option of CSVDYNL

REQUEST=UPDATE updates jobs or address spaces to use the current LNKLST set. For further information, see the topic "Removing or Compressing a Dataset in an Active LNKLST Set” in *z/OS MVS Initialization and Tuning Reference*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	SAF authority to the FACILITY class. If SAF is not available, or has no pertinent information, then the caller must be supervisor state or PKM 0-7, or PSW key 0-7, or APF-authorized. The SAF entity name and authorization level applied is UPDATE authority to FACILITY class entity CSVDYNL.UPDATE.LNKLST
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
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

The caller should include the CSVDLAA macro to get equate symbols for the return and reason codes.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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 if GPR15 is not 0

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 CSVDYNL macro is written as follows:

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

Syntax	Description
ƀ	One or more blanks must follow CSVDYNL.
REQUEST=UPDATE	
,WHICHAS=HOME	**Default:** WHICHAS=HOME
,WHICHAS=SPECIFIED	
,JOBNAME=*jobname*	*jobname:* RS-type address or address in register (2) - (12).
,ASID=*asid*	*asid:* RS-type address or address in register (2) - (12).
,DELAY=*d*	*d:* RS-type address or address in register (2) - (12).
,DELAY=NO_DELAY	**Default:** DELAY=NO_DELAY
,RETCODE=*retcode*	*retcode:* RS-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode:* RS-type address or register (2) - (12).
,PLISTVER=IMPLIED_VERSION	**Default:** PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=0	
,MF=S	**Default:** MF=S
,MF=(L,*list addr*)	*list addr:* RS-type address or register (1) - (12).
,MF=(L,*list addr,attr*)	
,MF=(L,*list addr*,0D)	
,MF=(E,*list addr*)	
,MF=(E,*list addr*,COMPLETE)	
,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 CSVDYNL macro invocation. The name must conform to the rules for an ordinary assembler language symbol.

REQUEST=UPDATE
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=UPDATE indicates to update the named address space so that subsequent operations will use the current LNKLST. Normally, a job continues to use the LNKLST set that was current when the job began. This function allows the job to use the current LNKLST set. Use this parameter with caution. Updating an address space while it is in the middle of fetching a module can cause the fetch to fail or to locate an incorrect copy of the module.

,WHICHAS=HOME
,WHICHAS=SPECIFIED
An optional parameter that indicates what address space is to be updated. The default is WHICHAS=HOME.

,WHICHAS=HOME
indicates the home address space.

,WHICHAS=SPECIFIED
indicates that the JOBNAME or ASID parameter specifies the address space. This option must be used with care, as modification of the LNKLST for any address space is dangerous unless the program knows that no program fetch processing is being done within that address space. IBM recommends that you do not use this option. It is provided primarily so that a program can fully duplicate the functions of the SETPROG LNKLST,UPDATE command.

,JOBNAME=*jobname*
,ASID=*asid*
When WHICHAS=SPECIFIED is specified, a required input parameter.

,JOBNAME=*jobname*
A parameter that contains the name of the job that is to be updated. All jobs of this name will be updated. If the jobname contains wildcard characters (? or *), the LNKLST will be updated for all jobs that match the provided pattern. The jobname used in performing the comparison is the name of the job for an initiated job; otherwise, the name of the address space.

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

,ASID=*asid*
A parameter that contains the ASID of the job.

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

DELAY=*d*
An optional input parameter that contains the value to delay the completion of the UPDATE operation in seconds. It must be a value in the range 0 to 255. The LNKLST will be updated immediately, but the processing for closing LNKLST data sets no longer in use and unallocating LNKLST data sets that are no longer in use is delayed by the given amount.

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

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

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

,RSNCODE=*rsncode*
An optional output parameter into which the reason code is to be copied from GPR 0.

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

,PLISTVER=IMPLIED_VERSION
,PLISTVER=MAX
,PLISTVER=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 CSVDYNL in the following order:

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

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

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

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

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

ABEND codes

None.

Return and reason codes

See “Return and reason codes” on page 468 for the return and reason codes.

Examples

See “Examples” on page 474 for an example.

REQUEST=QUERYDYN option of CSVDYNL

REQUEST=QUERYDYN queries to determine if the DEFINE, ADD, DELETE, UNDEFINE, TEST, and UPDATE functions are available.

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:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	The caller must not be locked.
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

The caller must include the CSVDLAA macro to get equates for the information returned via the DYNFUNC parameter.

Restrictions

The caller must not have functional recovery routines (FRRs) established.

Input register information

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

Before issuing the CSVDYNL 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

0-1
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

None.

Syntax

The CSVDYNL macro is written as follows:

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

Syntax	Description
CSVDYNL	
ƀ	One or more blanks must follow CSVDYNL.
REQUEST=QUERYDYN	
,DYNFUNC=*dynfunc*	*dynfunc:* RS-type address or address in register (2) - (12).

Parameters

The parameters are explained as follows:

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

REQUEST=QUERYDYN
A required parameter. REQUEST is required even when MF=(E,label,NOCHECK) is specified. REQUEST=QUERYDYN indicates to return an indication of whether LNKLST sets can be defined or changed. Those functions are available if the required DFSMS support is present. The RETCODE, RSNCODE, and MF keys cannot be specified.

,DYNFUNC=*dynfunc*
A required output parameter that will contain the availability of the DEFINE, ADD, DELETE, UNDEFINE, TEST, and UPDATE functions. If 0 (symbol CsvdynlDynNotAvailable in mapping macro CSVDLAA), the functions are not available. If 1 (symbol CsvdynlDynAvailable), the functions are available.

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

ABEND codes

None.

Return codes

None.

Examples

None.

Return and reason codes

When the CSVDYNL 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 CSVDLAA 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 48. Return and Reason Codes for the CSVDYNL Macro

Return Code	Reason Code	Equate Symbol Meaning and Action
0	—	**Equate Symbol**: CsvdynlRc_OK **Meaning**: CSVDYNL request successful. **DEFINE** **Meaning**: LNKLST set defined. **Action**: None required. **ADD** **Meaning**: Data set added to LNKLST set. **Action**: None required. **DELETE** **Meaning**: Data set removed from LNKLST set. **Action**: None required. **UNDEFINE** **Meaning**: LNKLST set removed. **Action**: None required. **TEST** **Meaning**: Routine was located using LNKLST set. **Action**: None required. **LIST** **Meaning**: All data returned. **Action**: None required. **UPDATE** **Meaning**: Specified job's LNKLST has been updated. **Action**: None required.
4	—	**Equate Symbol**: CsvdynlRc_Warn **Meaning**: Warning **Action**: Refer to the action provided with the specific reason code.
4	xxxx0402	**Equate Symbol**: CsvdynlRsnRoutineNotFound **Meaning**: For TEST request, routine was not found **Action**: Change the LNKLST set to contain the data set in which the requested routine is located.
4	xxxx0403	**Equate Symbol**: CsvdynlRsnNotAllDataReturned **Meaning**: For LIST request, not all data was returned because the answer area is not big enough. Answer area field DLAAHTLEN indicates how much space is currently required. **Action**: Allocate a larger area and request the function again.
4	xxxx0406	**Equate Symbol**: CsvdynlRsnNoMatchingJob **Meaning**: For UPDATE request, no matching job was found. **Action**: Make sure that you specified the proper jobname and ASID parameters.
8	—	**Equate Symbol**: CsvdynlRc_InvParm **Meaning**: CSVDYNL request specifies invalid parameters. **Action**: Refer to the action provided with the specific reason code.
8	xxxx0801	**Equate Symbol**: CsvdynlRsnBadParmlist **Meaning**: Unable to access parameter list. **Action**: Check for possible storage overlay.

Table 48. Return and Reason Codes for the CSVDYNL Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0802	**Equate Symbol**: CsvdynlRsnSrbMode **Meaning**: SRB mode. **Action**: Avoid requesting this function in SRB mode.
8	xxxx0803	**Equate Symbol**: CsvdynlRsnNotEnabled **Meaning**: Not Enabled. **Action**: Avoid requesting this function while not enabled.
8	xxxx0804	**Equate Symbol**: CsvdynlRsnNotAuthorized **Meaning**: Not authorized. **Action**: Request this function only when you have the proper authority.
8	xxxx0805	**Equate Symbol**: CsvdynlRsnHomeNotPrimary **Meaning**: Home address space different than primary address space. **Action**: Avoid requesting this function in this environment.
8	xxxx0806	**Equate Symbol**: CsvdynlRsnBadAnsareaALET **Meaning**: Bad answer area ALET. **Action**: Make sure that the ALET associated with the answer area is valid. The access register might not have been set up correctly.
8	xxxx0807	**Equate Symbol**: CsvdynlRsnBadAnsarea **Meaning**: Error accessing answer area. **Action**: Make sure that the provided answer area is valid.
8	xxxx0808	**Equate Symbol**: CsvdynlRsnBadAnslen **Meaning**: LIST - AnsLen is less than size of the header area. **Action**: Provide a larger answer area (as indicated by the ANSLEN parameter).
8	xxxx0809	**Equate Symbol**: CsvdynlRsnBadRequestType **Meaning**: Request type is not valid. **Action**: Check for possible storage overlay of the parameter list.
8	xxxx080A	**Equate Symbol**: CsvdynlRsnBadEstaex **Meaning**: Unable to establish ESTAEX. "xxxx" contains ESTAE(X) return code. **Action**: Refer to documentation for ESTAEX return code "xxxx".
8	xxxx080B	**Equate Symbol**: CsvdynlRsnReservedNot0 **Meaning**: Reserved field not 0. **Action**: Check for possible storage overlay of the parameter list.
8	xxxx080D	**Equate Symbol**: CsvdynlRsnBadParmlistALET **Meaning**: Bad parmlist ALET. **Action**: Make sure that the ALET of the parameter list is valid. The access register might not have been set up correctly.
8	xxxx080E	**Equate Symbol**: CsvdynlRsnBadVersion **Meaning**: Bad version number. **Action**: Check for possible storage overlay of the parameter list.
8	xxxx080F	**Equate Symbol**: CsvdynlRsnLocked **Meaning**: Locked **Action**: Avoid requesting this function in this environment.

Table 48. Return and Reason Codes for the CSVDYNL Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0815	**Equate Symbol**: CsvdynlRsnBadDsnameArea **Meaning**: Unable to access data set name. **Action**: Make sure that the DSNAME area is valid.
8	xxxx0816	**Equate Symbol**: CsvdynlRsnBadAfterDsnameArea **Meaning**: Unable to access AFTERDSNAME area. **Action**: Make sure that the AFTERDSNAME area is valid.
8	xxxx0818	**Equate Symbol**: CsvdynlRsnBadOpen **Meaning**: Unable to open specified data set. The Probdsname area is filled in with the name of the data set. **Action**: Make sure that you specified the proper data set and that it is partitioned and can be located by the system.
8	xxxx0819	**Equate Symbol**: CsvdynlRsnLnklstSetNotFound **Meaning**: LNKLST set name not found (for ADD, DELETE, UNDEFINE) **Action**: Make sure that you specified the proper LNKLST set name.
8	xxxx081C	**Equate Symbol**: CsvdynlRsnDatasetNotFound **Meaning**: For Delete request, data set was not associated with the LNKLST set. **Action**: Make sure that you specified the proper data set and LNKLST set names.
8	xxxx0820	**Equate Symbol**: CsvdynlRsnBadDsnameALET **Meaning**: Bad dsname ALET. **Action**: Make sure that the ALET of the DSNAME area is valid. The access register might not have been set up correctly.
8	xxxx0821	**Equate Symbol**: CsvdynlRsnBadAfterDsnameALET **Meaning**: Bad afterdsname ALET. **Action**: Make sure that the ALET of the AFTERDSNAME area is valid. The access register might not have been set up correctly.
8	xxxx0822	**Equate Symbol**: CsvdynlRsnBadLnklstName **Meaning**: Bad lnklstname - first character is 0 or blank. **Action**: Provide a valid LNKLST set name.
8	xxxx0823	**Equate Symbol**: CsvdynlRsnBadDsname **Meaning**: Bad DSNAME - first character is 0 or blank. **Action**: Provide a valid data set name.
8	xxxx0824	**Equate Symbol**: CsvdynlRsnBadAfterDsname **Meaning**: Bad AFTERDSNAME - first character is 0 or blank. **Action**: Provide a valid data set name.
8	xxxx0829	**Equate Symbol**: CsvdynlRsnBadAlloc **Meaning**: Unable to allocate data set. The Probdsname area is filled in with the name of the data set. **Action**: Make sure that you specified the proper data set and that it is partitioned and can be located by the system.
8	xxxx082B	**Equate Symbol**: CsvdynlRsnFunctionNotAvailableError **Meaning**: Dynamic allocation is not available or request issued during NIP. **Action**: Avoid requesting the function in an environment where dynamic allocation is not available. Avoid requesting the function until the IPL completes.

Table 48. Return and Reason Codes for the CSVDYNL Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx0831	**Equate Symbol**: CsvdynlRsnReservedName **Meaning**: Reserved name used. **Action**: Avoid specifying reserved names "CURRENT" and "IPL" when defining, adding to, deleting from, or undefining a LNKLST set.
8	xxxx0832	**Equate Symbol**: CsvdynlRsnNoJobASID **Meaning**: No jobname or ASID **Action**: Specify a jobname with the first character being non-zero and non-blank or specify a non-zero ASID for UPDATE.
8	xxxx0833	**Equate Symbol**: CsvdynlRsnAddSysdsn **Meaning**: Attempt to ADD after system data set. **Action**: Do not specify the LINKLIB, MIGLIB, CSSLIB, SIEALNKE, or SIEAMIGE data set via the AFTERDSNAME parameter. Specify POS=TOP if you want your data set to be in the first available position.
8	xxxx0834	**Equate Symbol**: CsvdynlRsnDeleteSysdsn **Meaning**: Attempt to DELETE system data set. **Action**: Do not specify the LINKLIB, MIGLIB, CSSLIB, SIEALNKE, or SIEAMIGE data set on REQUEST=DELETE. You cannot remove those data sets from the LNKLST.
8	xxxx0835	**Equate Symbol**: CsvdynlRsnNoCopyFrom **Meaning**: COPYFROM LNKLST set does not exist. **Action**: Be sure that you specified the proper LNKLST set name.
8	xxxx0836	**Equate Symbol**: CsvdynlRsnAlreadyExists **Meaning**: For DEFINE request, LNKLST set already exists. For ADD request, data set was already associated with this LNKLST set. **Action**: Make sure that you specified the proper data set and LNKLST set names.
8	xxxx0837	**Equate Symbol**: CsvdynlRsnNoModname **Meaning**: For TEST request, no module name was provided. **Action**: Make sure that you specified a module name that began other than with a blank or hexadecimal zeroes.
8	xxxx0839	**Equate Symbol**: CsvdynlRsnConcatFull **Meaning**: For ADD request, concatenation is full. The Probdsname area is filled in with the name of the data set at which the concatenation became full. **Action**: Remove data sets from the LNKLST set, or change the data sets to be PDSEs.
8	xxxx083A	**Equate Symbol**: CsvdynlRsnBadProbDsnameArea **Meaning**: For ADD or TEST request, unable to store into the PROBDSNAME area. **Action**: Make sure that the PROBDSNAME area is valid.
8	xxxx083B	**Equate Symbol**: CsvdynlRsnBadProbDsnameALET **Meaning**: For ADD or TEST request, ALET of ProbDsname is not acceptable. **Action**: Make sure that the ALET of the PROBDSNAME area is valid. The access register might not have been set up correctly.

Table 48. Return and Reason Codes for the CSVDYNL Macro (continued)

Return Code	Reason Code	Equate Symbol Meaning and Action
8	xxxx083C	**Equate Symbol**: CsvdynlRsnNotPartitioned **Meaning**: For ADD or TEST request, data set is not partitioned. The Probdsname area is filled in with the name of the data set. **Action**: Make sure that you specified the proper data set.
8	xxxx083D	**Equate Symbol**: CsvdynlRsnBadVolid **Meaning**: For ADD or TEST request, the provided volume ID does not match the volume ID found when the data set was looked up in the catalog. The Probdsname area is filled in with the name of the data set. **Action**: Make sure that you specified the proper volume ID. Make sure that the data set name in the catalog is cataloged to that volume. If the data set has been moved from one volume to another, then remove the data set from the LNKLST set, and add it back if the current volume is correct.
8	xxxx083E	**Equate Symbol**: CsvdynlRsnMultiVolume **Meaning**: Multi-Volume data set. **Action**: Avoid using a "multi-volume" data set within the LNKLST.
8	xxxx083F	**Equate Symbol**: CsvdynlRsnMissingSysdsn **Meaning**: The LNKLST set being tested does not contain at least one of SYS1.LINKLIB, SYS1.MIGLIB, and SYS1.CSSLIB. This should only occur if you used the SYSLIB statement of PROGxx. **Action**: Check with the system programmer.
8	xxxx0840	**Equate Symbol**: CsvdynlRsnUndefineCurrent **Meaning**: Cannot undefine the current LNKLST set. **Action**: Make sure that you specified the proper LNKLST set.
8	xxxx0841	**Equate Symbol**: CsvdynlRsnBadFoundDsnameArea **Meaning**: For TEST request, unable to store into the FOUNDDSNAME area. **Action**: Make sure that the FOUNDDSNAME area is valid.
8	xxxx0842	**Equate Symbol**: CsvdynlRsnBadFoundDsnameALET **Meaning**: For TEST request, ALET of FOUNDDSNAME is not acceptable. **Action**: Make sure that the ALET of the FOUNDDSNAME area is valid. The access register might not have been set up correctly.
8	xxxx0843	**Equate Symbol**: CsvdynlRsnBadSMS **Meaning**: For ADD or TEST request, the SMS status of the data set has changed. Either it is now SMS-managed but had not been, or is not SMS-managed but had been. The Probdsname area is filled in with the name of the data set. **Action**: Remove the data set from the LNKLST set. Add it back if the current SMS status of the data set is correct.
C	—	**Equate Symbol**: CsvdynlRc_Env **Meaning**: Environmental error **Action**: Refer to the action provided with the specific reason code.
C	xxxx0C01	**Equate Symbol**: CsvdynlRsnFunctionNotAvailable **Meaning**: Meaning: Function is not available. **Action**: Avoid requesting this function until system initialization is complete.
C	xxxx0C02	**Equate Symbol**: CsvdynlRsnNoStorage **Meaning**: No storage is available to complete the request. **Action**: Contact your system programmer. There is a common storage shortage.

Table 48. Return and Reason Codes for the CSVDYNL Macro (continued)		
Return Code	**Reason Code**	**Equate Symbol Meaning and Action**
C	xxxx0C03	**Equate Symbol**: CsvdynlRsnChangeInUse **Meaning**: Cannot update an in-use LNKLST set. **Action**: Make sure that you specified the proper LNKLST set. You will have to wait if you need to modify a LNKLST set that is in use until it is no longer the active set. You can alternately create a new LNKLST set, initialized from the active one. You can use the UPDATE request to update users of prior LNKLST sets to be using the current LNKLST set, thus letting you update that prior set.
C	xxxx0C04	**Equate Symbol**: CsvdynlRsnUndefineUsers **Meaning**: Cannot undefine a LNKLST set that is in use. **Action**: Make sure that you specified the proper LNKLST set. You can use DISPLAY PROG,LINK to display the current users of the LNKLST set. You can consider using CSVDYNEX REQUEST=UPDATE to update the current users to the current LNKLST set, but refer to that operation for appropriate caveats.
C	xxxx0C05	**Equate Symbol**: CsvdynlRsnActivateNoENF **Meaning**: Could not issue ENF signal for ACTIVATE request. **Action**: Contact your system programmer.
C	xxxx0C06	**Equate Symbol**: CsvdynlRsnUndefineLLA **Meaning**: Cannot undefine a LNKLST set that LLA is using to manage the LNKLST. **Action**: Make sure that you specified the proper LNKLST set. You can use DISPLAY PROG,LINK to display the current users of the LNKLST set. You can consider using CSVDYNEX REQUEST=UPDATE to update the current users to the current LNKLST set, but refer to that operation for appropriate caveats.
10	—	**Equate Symbol**: CsvdynlRC_CompError **Meaning**: Unexpected failure. **Action**: Refer to the action provided with the specific reason code.
10	xxxx1001	**Equate Symbol**: CsvdynlRsnCompError **Meaning**: Unexpected failure. The state of the request is unpredictable. **Action**: Contact your system programmer.

Examples

Example 1:

Operation 1

1. Define a LNKLST set.
2. Add data sets to that LNKLST set.
3. Delete a data set from the LNKLST set.
4. Test the LNKLST set for the presence of a particular module.
5. Undefine the LNKLST set.

The code is as follows.

```
************************************************************
* Define LNKLST set MYLNKLST_SET.                          *
* Copy it from the current LNKLST set.                     *
************************************************************
         CSVDYNL REQUEST=DEFINE,LNKLSTNAME=LLS,             *
               COPYFROM==CL16'CURRENT',                     *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
```

```
*
* Place code to check return/reason codes here
*
*********************************************************************
* Add data set SYS1.MYDS1 at the top of the user-specified           *
* data sets (but after the system-defined data sets                  *
* SYS1.LINKLIB, SYS1.MIGLIB, SYS1.CSSLIB,SYS1.SIEALNKE, SYS1.SIEAMIGE)*
*********************************************************************
         CSVDYNL REQUEST=ADD,LNKLSTNAME=LLS,                *
               DSNAME=LDS1,POS=TOP,                         *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
*
* Place code to check return/reason codes here
*
************************************************************
* Add data set SYS1.MYDS2 at the bottom of the             *
* user-specified data sets                                 *
************************************************************
         CSVDYNL REQUEST=ADD,LNKLSTNAME=LLS,                *
               DSNAME=LDS2,POS=BOTTOM,                      *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
*
* Place code to check return/reason codes here
*
************************************************************
* Add data set SYS1.MYDS3 after SYS1.MYDS1                 *
************************************************************
         CSVDYNL REQUEST=ADD,LNKLSTNAME=LLS,                *
               DSNAME=LDS3,                                 *
               POS=AFTER,AFTERDSNAME=LDS1,                  *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
*
* Place code to check return/reason codes here
*
************************************************************
* Delete data set SYS1.DONTWANT from the LNKLST set        *
************************************************************
         CSVDYNL REQUEST=DELETE,LNKLSTNAME=LLS,             *
               DSNAME=LDONTWANT,                            *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
*
* Place code to check return/reason codes here
*
************************************************************
* Test the LNKLST set to see if routine MYMODULE can be    *
* found                                                    *
************************************************************
         CSVDYNL REQUEST=TEST,LNKLSTNAME=LLS,               *
               MODNAME=LMOD,                                *
               FOUNDDSNAME=LDSF,                            *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
*
* Place code to check return/reason codes here.
* Note that when the return code is 0, the name of the data
* set which the routine was found is placed into LDSF
*
************************************************************
* Undefine the LNKLST set                                  *
************************************************************
         CSVDYNL REQUEST=UNDEFINE,LNKLSTNAME=LLS,           *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
*
* Place code to check return/reason codes here.
*
LLS      DC    CL16'MYLNKLST_SET'
LMOD     DC    CL8'MYMODULE'
LDS1     DC    CL44'SYS1.MYDS1'
LDS2     DC    CL44'SYS1.MYDS2'
LDS3     DC    CL44'SYS1.MYDS3'
LDONTWANT DC   CL44'SYS1.DONTWANT'
         CSVDLAA              Return code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
```

```
LDSF     DS    CL44
         CSVDYNL MF=(L,DYNLL)
```

Example 2

Operation 2

- Update the address space to use the current LNKLST set.

The code is as follows.

```
*************************************************************
* Update this address space to be using the current LNKLST  *
* set.  This would generally be done in response to         *
* receiving an ENF signal indicating that a new LNKLST set  *
* had been made available, and would be done at a time when *
* the program had reason to believe that the address space  *
* was not in the middle of fetching (e.g., LINK, LOAD) a    *
* routine.                                                  *
*************************************************************
         CSVDYNL REQUEST=UPDATE,                            *
               WHICHAS=HOME,                                *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
*
* Place code to check return/reason codes here
*
         CSVDLAA               Return code information
DYNAREA  DSECT
LRETCODE DS    F
LRSNCODE DS    F
         CSVDYNL MF=(L,DYNLL)
```

Example 3

Operation 3

- Retrieve information about all of the LNKLST sets.

The code is as follows.

```
         L     2,=AL4(INITDLAA)  Initial answer area size
         ST    2,SIZEDLAA      Save it
         GETMAIN RU,LV=(2)     Allocate the answer area
         ST    1,DLAA@         Save address of answer area
LAB1     DS    0H
         L     4,DLAA@         Address of answer area
         CSVDYNL REQUEST=LIST,ANSAREA=(4),ANSLEN=SIZEDLAA,   *
               RETCODE=LRETCODE,RSNCODE=LRSNCODE,           *
               MF=(E,DYNLL)
         CLC   LRETCODE(4),=AL4(CSVDYNLRC_WARN) Warning?
         BNE   LAB2            No, request OK or error
*                              Yes, not enough room
         LR    3,2             Save current size
         L     2,DLAAHTLEN-DLAAHDR(4) Get required size
         FREEMAIN RU,A=(4),LV=(3) Release old area
         ST    2,SIZEDLAA      Save it
         GETMAIN RU,LV=(2)     Allocate new area
         ST    1,DLAA@         Save address of answer area
         B     LAB1            Retry List operation
LAB2     DS    0H
         CLC   LRETCODE(4),=AL4(CSVDYNLRC_OK) Success?
         BNE   LAB3            No, error
*************************************************************
*                                                           *
* Process information in answer area when RC=0              *
*                                                           *
*************************************************************
         USING DLAAHDR,4       DLAAHDR DSECT
         L     5,DLAAH#LS      Find how many DLAALS entries
         LTR   5,5             Are there any entries
         BZ    LAB4            No, join common path
         L     4,DLAAHFIRSTLS@ Get first entry
         USING DLAALS,4        DLAALS DSECT
LAB5     DS    0H              DLAALS loop
*
```

```
* Put code to process information contained in DLAALS here
*
         LH    7,DLAALS#DS     Get number of DLAADS entries
         N     7,CLEAR0TO15    Clear bits 0 to 15
*                              Are there any data sets
         BZ    LAB7            No, move to end of LS loop
         L     6,DLAALSFIRSTDS@ Get first DLAADS
LAB6     DS    0H              DLAADS loop
         USING DLAADS,6        DLAADS DSECT
*
* Put code to process information contained in DLAADS here
*
         L     6,DLAADSNEXT@   Get next DLAADS
         DROP  6               DLAADS DSECT
         BCT   7,LAB6          Continue while there are more
LAB7     DS    0H              Bottom of DLAALS loop
         L     4,DLAALSNEXT@   Get next DLAALS
         BCT   5,LAB5          Continue while there are more
         B     LAB4            Skip error case
LAB3     DS    0H              Error return
*
*  Process error case
*
LAB4     DS    0H              Common path
         L     2,SIZEDLAA      Get size of area
         L     4,DLAA@         Get address of area
         FREEMAIN RU,A=(4),LV=(2) Release area
         ...
CLEAR0TO15 DC  A(X'0000FFFF')  Mask to clear bits 0-15
         CSVDLAA               LIST answer area, return codes
DSNLEN   EQU   50*DLAADS_LEN   Room for 50 data sets' info
LSLEN    EQU   3*DLAALS_LEN    Room for 3 LNKLST sets' info
INITDLAA EQU   DLAAHDR_LEN+DSNLEN+LSLEN  Initial ansarea size
DYNAREA  DSECT
DLAA@    DS    A               Address of answer area
SIZEDLAA DS    F               Size of answer area
TEMPSIZE DS    F               Temporary
LRETCODE DS    F               Return code
LRSNCODE DS    F               Reason code
         CSVDYNL MF=(L,DYNLL)
```

Chapter 40. CTRACE — Define a user application to the component trace service

Description

The CTRACE macro defines a user application to the component trace service also referred to as "component trace". An application using component trace must have an installation-written start/stop exit routine to start, stop, or modify tracing for the application. Once you define the application to component trace:

- The application's start/stop routine can get control through a parmlib member specified on the PARM parameter on CTRACE DEFINE. If the parmlib member contains trace options that tell the system to turn on the trace, the system passes control to the start/stop routine without operator intervention. See *z/OS MVS Initialization and Tuning Reference* for information about setting up one or more CTncccxx parmlib members.
- The operator can pass control to the start/stop routine, and can specify trace options through operator commands, with or without specifying parmlib members. The operator can also display the status of the application's trace. See *z/OS MVS System Commands* for the command syntax and parameter descriptions for the TRACE and DISPLAY commands.

Before the application ends, it should use the CTRACE macro to delete itself from component trace. If a component trace is active, the CTRACE DELETE macro calls the start/stop exit routine to clean up resources and stop tracing. Deleting the trace prevents the system from reporting an inactive trace as active when the operator requests a display of the application's trace.

For information on writing the start/stop routine and for additional information about using the CTRACE macro, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Once your application creates trace entries and externalizes them, either in a dump or through the component trace external writer, use the interactive problem control system (IPCS) to format the trace data. See *z/OS MVS IPCS Commands* and *z/OS MVS IPCS Customization* for details.

For an understanding of tracing in general, and for details on planning to use component trace to trace an application, see *z/OS Problem Management*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and PSW key 0
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

An application using component trace must have an installation-written start/stop exit routine to start, stop, or modify tracing for the application.

Restrictions

None.

Register information

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

The caller must ensure that register 13 points to a standard 72-byte save area.

When control returns to the caller, the GPRs contain:

Register
Contents

0
Used as a work register by the system if GPR 15 contains 0 or 4; otherwise, GPR 0 contains a reason code.

1
Used as a work register by the system

2-12
Unchanged.

13
Contains the address of a standard save area.

14
Used as a work register by the system

15
Return code

Performance implications

All component traces use system resources and have some impact on performance. You should evaluate the impact of using either a single trace or multiple traces and determine which trace provides the information you need with the least effect on performance. For information about multiple and single traces, see *z/OS Problem Management*

Syntax

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

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

Syntax	Description
ƀ	One or more blanks must follow CTRACE.
DEFINE	
DELETE	
,NAME=*name*	*name*: RX-type address or register (2) - (12).
,STARTNAM=*sname*	*sname*: RX-type address or register (2)-(12).
,DISPNAM=*dname*	*dname*: RX-type address or register (2)-(12).
,PARM=*parm*	*parm*: RX-type address or register (2) - (12).
,PARM=NOPARM	**Default:** PARM=NOPARM.
,ASIDS=NO	**Default:** ASIDS=NO.
,ASIDS=YES	
,JOBS=NO	**Default:** JOBS=NO.
,JOBS=YES	
,MINOPS=*options*	*options*: RX-type address.
,MINOPS=NONE	**Default:** MINOPS=NONE.
,ON=MIN	**Default:** ON=MIN
,ON=NOTMIN	
,MOD=NO	**Default:** MOD=NO.
,MOD=YES	
,FMTTAB=*fmtabs*	*fmtabs*: RX-type address or register (2) - (12).
,FMTTAB=NONE	**Default:** FMTTAB=NONE.
,USERDATA=*userdata*	*userdata*: RX-type address or register (2) - (12).
,USERDATA=NOUSERDATA	**Default:** USERDATA=NOUSERDATA

Syntax	Description
,HEAD=NO	**Default:** HEAD=NO.
,HEAD=YES	
,HEADOPTS=NO	**Default:** HEADOPTS=NO.
,HEADOPTS=YES	
,SUB=*subname*	*subname*: RX-type address or register (2) - (12).
,SUB=NOSUB	**Default:** SUB=NOSUB.
,LIKEHEAD=NO	**Default:** LIKEHEAD=NO.
,LIKEHEAD=YES	
,MANYSUBS=NO	**Default:** MANYSUBS=NO.
,MANYSUBS=YES	
,DELSUBS	**Default:** DELSUBS
,IFNOSUBS	
,BUFFER=NO	**Default:** BUFFER=NO.
,BUFFER=YES	
,BUFDEFIN=NO	**Default:** BUFDEFIN=NO.
,BUFDEFIN=YES	
,BUFMIN=*minsize*	*minsize*: Minimum buffer size.
	Default: BUFMIN=1024.
,BUFMAX=*maxsize*	*maxsize*: Maximum buffer size.
	Default: BUFMAX=2147483647.
,BUFDFLT=*dfltsize*	*dfltsize*: RX-type address or register (2) - (12).
,BUFDFLT=NODFLT	**Default:** BUFDFLT=NODFLT.
,WTR=NO	**Default:** WTR=NO.

Syntax	Description
,WTR=YES	
,WTRMODE=PAGEABLE	
,WTRMODE=DREF	
,WTRMODE=FIXED	
,SSRC=*ss_retcode*	*ss_retcode*: RX-type address or register (2) - (12).
,SSRSNC=*ss_rsncode*	*ss_rsncode*: RX-type address or register (2) - (12).
,RC=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RX-type address or register (2) - (12).
,COM=*comment*	*comment*: A comment string.
,COM=NULL	**Default:** NULL.
,MF=(S)	**Default:** MF=(S)

Parameters

The parameters are explained as follows:

DEFINE | DELETE
The required parameter that either defines the application to or deletes the application from component trace.

When you specify DEFINE:

- You must specify NAME.
- You cannot specify DELSUBS or IFNOSUBS.

All other parameters are optional with DEFINE.

When an application is using multiple traces, the application must issue CTRACE DEFINE separately for each trace.

When you specify DELETE:

- You must specify NAME.
- SUB, DELSUBS, IFNOSUBS, RSNCODE, RC, COM, and MF are optional parameters.

You cannot specify any other parameters with DELETE.

,NAME=*name*
The required parameter that specifies the name of the application or head node to be defined or deleted. The name must begin with an alphabetic or national character and can contain up to eight

alphanumeric or national characters. The first three letters must not be *SYS* because *SYS* is reserved for IBM use. NAME is required for both DEFINE and DELETE.

The operator uses this name on the TRACE CT command (COMP parameter), and IPCS uses this name on the CTRACE subcommand (COMP parameter).

,STARTNAM=*sname*
Specifies the name of the application's start/stop exit routine. This routine receives control to start, stop, or modify tracing for the application. The application must have at least one start/stop routine. In the case of multiple traces, the application might have a start/stop routine for the head node and for each sub-level trace.

This routine must be a reentrant load module in the LNKLST or LPA.

You must code STARTNAM whenever you code CTRACE DEFINE, unless you code HEADOPTS=NO with HEAD=YES. If you code HEADOPTS=NO with HEAD=YES, STARTNAM is not valid.

,DISPNAM=*dname*
Specifies the name of the application's display exit routine. This routine receives control to provide status information about the component trace for the DISPLAY TRACE operator command.

This routine must be a reentrant load module in the LNKLST or LPA.

You can code DISPNAM whenever you code CTRACE DEFINE, unless you code HEADOPTS=NO with HEAD=YES. If you code HEADOPTS=NO with HEAD=YES, DISPNAM is not valid.

,PARM=*parm* | NOPARM
Specifies the name of a parmlib member that contains the options to be used for tracing. A parmlib member specified on the CTRACE macro can contain tracing options for only one trace. Consult *z/OS MVS Initialization and Tuning Reference* for information on how to set up one or more component trace (CTncccxx) parmlib members.

The default is PARM=NOPARM.

,ASIDS=YES | NO
Allows you to request trace filtering by ASIDs. A single trace that uses multiple address spaces can use this parameter as a filter to ensure that a trace is done only in certain address spaces. With ASIDS=YES, you can specify up to 16 ASIDs in a parmlib member on the PARM parameter, or the operator can specify up to 16 ASIDs. With ASIDS=NO, which is the default, neither you nor the operator can request trace filtering by ASIDs.

,JOBS=YES | NO
Allows you to request trace filtering by JOBNAMEs. With JOBS=YES, you can specify up to 16 JOBNAMEs in a parmlib member on the PARM parameter, or the operator can specify up to 16 JOBNAMEs. With JOBS=NO, which is the default, neither you nor the operator can request trace filtering by JOBNAMEs.

,MINOPS=*options* | NONE
Specifies a list of user-defined options that are in effect when the trace is off or no other options are specified. These options cannot be turned off by the operator. The character string for the options list must not exceed 255 bytes. The individual options must be separated by commas. See *z/OS Problem Management* for information about planning and setting up user-defined options for your application. The default is MINOPS=NONE.

,ON=MIN | NOTMIN
This parameter is an optional keyword, which indicates whether the "ON" state (with no options) differs from the "MIN" state. If you code ON=MIN, you need to specify different options to get different behavior of the CTRACE than when it is started in the minimum state. A correct value for this option helps the processing of the DISPLAY TRACE,CT command and CHECK(IBMCTRACE,CTRACE_DEFAULT_OR_MIN). The default is ON=MIN.

When you specify ON=MIN, the ON state matches the MIN state if no options are provided.

When you specify ON=NOTMIN, the ON state does not match the MIN state.

,MOD=NO | YES
Specifies whether an application's trace must be stopped before changes are made to the application's tracing options. If you code MOD=YES, you allow the tracing options to be modified without stopping the trace. The default is MOD=NO.

,FMTTAB=*fmtabs* | NONE
Specifies the name of the load module that contains the CTRACE format table for the application. Use the ITTFMTB macro, described in *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG* to create this CTRACE format table. See *z/OS MVS IPCS Customization* for further details about how to create a CTRACE format table.

The default,FMTTAB=NONE, specifies that IPCS is not to format the trace.

Note: Specifying either the FMTTAB=*fmtabs* and USERDATA=*userdata* parameters cause information to be placed in the SQA. If SQA is not dumped, the information supplied by CTRACE DEFINE will not enable formatting to proceed. You can supply a CTRACE statement in parmlib member BLSCUSER to enable formatting to commence in this situation, but the buffer finds routine supplied by your component is supplied with *userdata* of zeros.

,USERDATA=*userdata* | NOUSERDATA
Specifies an optional 16 byte input/output area that is used by the application to contain any data that it wants to associate with this trace. When control is passed to the trace's start/stop routine, this user data field is passed in the CTSS. Similarly, the user data field is passed in the CTXI to the trace's IPCS exit routines. Suggested uses include placing the address (and optionally, the ALET) of either of the following into this field:

- The application's control information
- The application's first trace buffer.

Note: Specifying either the FMTTAB=*fmtabs* and USERDATA=*userdata* parameters cause information to be placed in the SQA. If SQA is not dumped, the information supplied by CTRACE DEFINE will not enable formatting to proceed. You can supply a CTRACE statement in parmlib member BLSCUSER to enable formatting to commence in this situation, but the buffer finds routine supplied by your component is supplied with *userdata* of zeros.

,HEAD=NO | YES
When using multiple traces, use this parameter to specify whether the trace you are defining is a head node. If you specify HEAD=YES, you can define sub-level traces (SUB parameter) on **subsequent** invocations of CTRACE DEFINE that share the options, attributes, and state of this head node.

If you specify both HEAD=YES and SUB=*sub* on the same invocation of CTRACE DEFINE, both of the following are true:

- The trace that you are defining is a head node for sub-level traces to be defined on **subsequent** invocations of CTRACE DEFINE.
- The trace that you are defining is a sub-level trace for a head node that was defined on a **subsequent** invocation of CTRACE DEFINE.

If you specify HEAD=NO (or take the default), you cannot define sub-level traces under this trace.

,HEADOPTS=NO | YES
Specifies that the application supports options for the head node that is being defined. This parameter is valid only if you specify HEAD=YES.

When you specify HEADOPTS=YES, you can define sub-level traces on subsequent invocations of CTRACE DEFINE with the LIKEHEAD parameter and those traces have the same options, attributes, and state as the HEAD. When any of the options or the state of the head are changed, all of the sub-level traces that are defined LIKEHEAD are also changed.

If you specify HEADOPTS=YES, you must specify a start/stop routine.

If you specify HEADOPTS=NO, you define a head node whose sub-level traces are independent of the head node. You will not be able to turn on the trace at the head node, but only at the sub-levels. Therefore, if HEADOPTS=NO, you cannot specify a start/stop routine.

HEADOPTS=NO is the default.

,SUB=*subname* | NOSUB
When using multiple traces, use this parameter to specify that the trace you are defining is a sub-level trace, and specifies the name of the sub-level trace. Specify a sub-level trace name in one of the following ways:

- A 1- to 18-character name that begins with a letter, A-Z, or the characters $, #, or @ and consists of a combination of the characters A-Z, 0-9, and the characters $, #, and @. If you enclose the name in quotation marks, you can also use lowercase letters, a-z, and the underscore character (_).
- The keyword ASID or JOBNAME. For example, for job ABC running in address space ASID(05), you can code either of the following for the same result:
  - SUB=JOBNAME(ABC)
  - SUB=ASID(05)

If you define multiple sub-level traces on one path of the trace structure, you can specify up to five sub-level trace names separated by periods; for example:

```
CTRACE DEFINE,NAME=APPLABC,SUB=ASID(13).FACILITY12.SUBA.SUBB.SUBC,...
```

,LIKEHEAD=NO | YES
When using multiple traces, use this parameter to specify whether the sub-level trace you are defining is to use the options, attributes, and state of its head node (defined on a previous invocation of the CTRACE macro.) The default is LIKEHEAD=NO.

You **cannot** specify LIKEHEAD=YES with the following:

- HEADOPTS=NO
- PARM=*parm*
- Any of the attributes keywords (ASIDS, JOBS, MINOPS, MOD, BUFFER, BUFMIN, BUFMAX, BUFDFLT, BUFDEFIN, and WTR).

,MANYSUBS=NO | YES
Specifies whether you expect the head node you are defining to have more than 15 sub-level traces directly associated with it. For example, a head node with NAME=APPLABC might have 20 sub-level traces that are specified on subsequent invocations of CTRACE DEFINE that are directly associated with APPLABC:

```
CTRACE DEFINE,NAME=APPLABC,SUB=SUB1,...
CTRACE DEFINE,NAME=APPLABC,SUB=SUB2,...
CTRACE DEFINE,NAME=APPLABC,SUB=SUB3,...
⋮
CTRACE DEFINE,NAME=APPLABC,SUB=SUB20,...
```

In this case, code MANYSUBS=YES when you define the head node.

The default is MANYSUBS=NO.

,DELSUBS
,IFNOSUBS
When using multiple traces, use this parameter to restrict which traces are deleted.

When you issue CTRACE DELETE to delete a trace, and you specify IFNOSUBS, the system does the following:

- If the trace is not a head node, the system deletes the trace.
- If the trace is a head node with no sub-level traces associated with it, the system deletes the trace.
- If the trace is a head node with sub-level traces, the system issues return code X'0C' with reason code X'2B' and does not delete the trace.

When you issue CTRACE DELETE to delete a trace, and you specify DELSUBS, the system deletes the trace. If the trace is a head node with sub-level traces associated with it, the system also deletes the sub-level traces. DELSUBS is the default.

,BUFFER=NO | YES
Restricts how you can specify the application's trace buffer size. If you code BUFFER=YES, you can specify the trace buffer size in a parmlib member on the CTRACE macro, or the operator can specify the trace buffer size.

If you code BUFFER=NO, which is the default:

- You cannot specify a trace buffer size in a parmlib member on the CTRACE macro.
- The operator cannot specify a trace buffer size.
- You cannot specify BUFDEFIN, BUFMIN, BUFMAX, or BUFDFLT.

,BUFDEFIN=NO | YES
If you code BUFDEFIN=YES, a buffer size can be specified only on a parmlib member with the PRESET option, or in a parmlib member that you specify on the PARM parameter of CTRACE DEFINE. If you specify BUFDEFIN=YES you must also specify BUFFER=YES. BUFDEFIN=NO is the default.

,BUFMIN=*minsize*
Specifies the minimum buffer size allowed. The default is 1024 bytes. If you specify less than the default, the default is used. If you specify BUFMIN, you must also specify BUFFER=YES.

,BUFMAX=*maxsize*
Specifies the maximum buffer size allowed. BUFMAX cannot be less than BUFMIN. The default is 2147483647 bytes. If you specify BUFMAX, you must also specify BUFFER=YES.

,BUFDFLT=*dfltsize* | NODFLT
Specifies the default buffer size to be used if you do not specify a buffer size in a parmlib member on the PARM parameter, or if the operator does not specify a buffer size. *dfltsize* cannot be less than BUFMIN or more than BUFMAX.

If you specify BUFDFLT=*dfltsize* and also specify a buffer size in a parmlib member or on an operator command, the *dfltsize* is overridden.

To specify BUFDFLT=*dfltsize*, you must also specify BUFFER=YES.

BUFDFLT=NODFLT is the default.

,WTR=NO | YES
Specifies whether the application's trace supports writing trace data to DASD or tape through the component trace external writer. If you specify WTR=YES, you must also specify WTRMODE.

The default, WTR=NO, means that an external writer is not used for this trace.

,WTRMODE=PAGEABLE | DREF | FIXED
Indicates the type of storage the application's trace buffers are in when you issue the CTRACEWR macro to write the trace buffers to DASD or tape. This parameter is required when you specify WTR=YES.

When you code WTRMODE=PAGEABLE, the application's trace buffers can be in either fixed, disabled reference (DREF), or pageable storage at the time you issue CTRACEWR.

Note: IBM recommends that you keep your trace buffers in pageable storage, which will not deplete your system's central storage.

When you code WTRMODE=DREF, the application's trace buffers can be in either fixed or DREF storage at the time you issue CTRACEWR. Regular page faults are not allowed.

When you code WTRMODE=FIXED, the application's trace buffers must be in fixed storage at the time you issue CTRACEWR.

,SSRC=*ss_retcode*
Specifies the name of a 4-byte output area to contain the return code set by the start/stop routine.

,SSRSNC=*ss_rsncode*
Specifies the name of a 4-byte output area to contain the reason code from the start/stop routine.

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

,RSNCODE=*rsncode*
Specifies the location where the system is to store the reason code. The reason code is also in GPR 0. CTRACE provides a reason code if the return code is other than 0 or 4.

,COM=*comment*
,COM=NULL
Allows you to include a comment string in the macro block comment before the macro invocation. If the comment contains any lowercase characters, it must be enclosed in quotation marks.

,MF=(S)
Specifies the standard form of the CTRACE macro.

ABEND codes

The following table identifies abend code and reason code combinations, and a description of what each means:

Table 49. Abend codes for the CTRACE Macro

Abend Code	Reason Code	Description
00D	00000101	For the CTRACE DEFINE macro, the parameter list version number is not correct.
00D	00000102	For the CTRACE DEFINE macro, the component name either does not begin with an alphabetic or national character, or it contains one or more characters that are not alphanumeric or national characters.
00D	00000301	The system found either nonzero values in the reserved fields or unused fields for the requested service in the CTRACE DEFINE macro parameter list.
00D	00000302	The system found either nonzero values in the reserved fields or unused fields for the requested service in the CTRACE DELETE macro parameter list.
00D	00000401	For the CTRACE macro, an incorrect service request was specified. Valid services are CTRACE DEFINE and CTRACE DELETE.
00D	00000501	For the CTRACE DEFINE macro, the length of minimum options string is greater than 256 bytes.

Return and reason codes

When control returns from CTRACE, GPR 15 (and *retcode*, if you coded RC) contains one of the following return codes. The third byte of GPR 0 (and *rsncode*, if you coded RSNCODE) might contain one of the following reason codes.

Note: The application should always check the return code from the CTRACEWR macro. A non-zero return code indicates that some data might have been lost in the next record output.

Table 50. Return and Reason Codes for the CTRACE Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00	None.	CTRACE was successful.
04	None.	CTRACE was unsuccessful. • For the DEFINE request, the application was already defined to component trace. • For the DELETE request, the application is not defined to component trace.
08	xxxx06xx	Insufficient storage for a DEFINE operation.
08	xxxx07xx	CTRACE could not establish a recovery environment.
0C	xxxx01xx	An attempt to define a SUB was made before a HEAD was defined.

Table 50. Return and Reason Codes for the CTRACE Macro (continued)

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
0C	xxxx02xx	The LIKEHEAD option was specified with other trace options.
0C	xxxx03xx	An attempt to define a SUB was made, but the previous level was not a HEAD.
0C	xxxx04xx	The specified parmlib member was not found.
0C	xxxx05xx	While attempting to read the specified parmlib member, an I/O error occurred.
0C	xxxx06xx	There is a syntax error in the specified parmlib member.
0C	xxxx07xx	LIKEHEAD=YES was specified, but no HEAD exists for a SUB to match. The head node was defined with HEADOPTS=NO.
0C	xxxx08xx	JOBNAME or ASID was specified as the SUB name, but the job or address space is not active.
0C	xxxx09xx	LIKEHEAD was specified in the parmlib member, but the HEAD had different attributes.
0C	xxxx0Axx	LIKEHEAD=YES was specified with the PARM keyword.
0C	xxxx0Bxx	HEADOPTS=NO was specified with the PARM keyword.
0C	xxxx0Cxx	LIKEHEAD=YES was specified with HEADOPTS=NO.
0C	xxxx0Dxx	STARTNAM is required for all defines except when HEADOPTS=NO.
0C	xxxx0E01	STARTNAM is not allowed when both HEADOPTS=NO and HEAD=YES are specified.
0C	xxxx0E02	DISPNAM is not allowed when both HEADOPTS=NO and HEAD=YES are specified.
0C	xxxx0Fxx	SUB cannot be specified in the parmlib member.
0C	xxxx10xx	PRESET(DELETE) cannot be specified in the parmlib member.
0C	xxxx11xx	The start/stop routine returned a nonzero return code.
0C	xxxx12xx	A buffer cannot be specified on this invocation of the CTRACE macro.
0C	xxxx13xx	The buffer size (BUFSIZE) specified in the parmlib member is not within the limits as defined by BUFMIN and BUFMAX.
0C	xxxx14xx	ASID filtering is not allowed.
0C	xxxx15xx	Jobname filtering is not allowed.
0C	xxxx16xx	The BUFMIN specified is greater than the BUFMAX.
0C	xxxx17xx	The BUFDFLT specified is less than the BUFMIN.
0C	xxxx18xx	The BUFDFLT specified is greater than the BUFMAX.
0C	xxxx19xx	The LOAD or LINK to the specified start/stop routine failed.
0C	xxxx1Axx	A specified sub-level trace name is not valid. Either the syntax is not correct, or more than five names were specified.
0C	xxxx1Bxx	A parmlib error was found.
0C	xxxx1Cxx	An ASID is not a valid hexadecimal number.
0C	xxxx1Dxx	The parmlib member, including comments, is too large.
0C	xxxx1Fxx	The parmlib member cannot be read.
0C	xxxx20xx	The dynamic allocation of a parmlib member failed.
0C	xxxx21xx	An ASID is longer than four characters.
0C	xxxx22xx	An ASID of zero is not valid.
0C	xxxx23xx	More than 16 ASIDs were specified.

Table 50. Return and Reason Codes for the CTRACE Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning**
0C	xxxx24xx	A jobname is longer than eight characters.
0C	xxxx25xx	More than 16 jobnames were specified.
0C	xxxx26xx	The buffer size specification is longer than five characters.
0C	xxxx27xx	The buffer size specification does not have K or M specified as the unit.
0C	xxxx28xx	The buffer size specified is not a valid decimal number.
0C	xxxx29xx	The options string is longer than 1024 characters.
0C	xxxx2Axx	The parmlib member name did not begin with the characters 'CT.'
0C	xxxx2Bxx	The DELETE failed because IFNOSUBS was specified and the trace had sub-level traces.
0C	xxxx2Cxx	The trace does not support using an external writer.
0C	xxxx2Dxx	The name of the external writer is not valid. A valid name is 1-7 characters in length, starting with an alphabetic or national character (A-Z, $, @, #) and containing alphanumeric or national characters (A-Z, 0-9, $, @, #).

Example

Define application APPXYZ to component trace as a head node, and allow sub-level traces to be defined with the same options, attributes, and state as the head node. Use a parmlib member to supply default options for the trace. The name of the head node's start/stop exit routine is APPXYZSS. The system is to store the return and reason codes from the start/stop routine in SSRC and SSRSNC. The system is to store the return and reason codes from the CTRACE macro in TCRC and TCRSN.

```
          CTRACE DEFINE,NAME=COMPNAM1,STARTNAM=STRTNAM1,                X
                HEADOPTS=YES,HEAD=YES,ASIDS=NO,JOBS=NO,                 X
                BUFFER=YES,BUFDEFIN=NO,BUFDFLT=5000,                    X
                SSRC=SSRC,SSRSNC=SSRSN,MANYSUBS=YES,                    X
                WTR=YES,WTRMODE=PAGEABLE,PARM=PARM1,MOD=YES,            X
                RC=TCRC,RSNCODE=TCRSN

          COMPNAM1 DC    CL8'APPXYZ  '  Component name
          STRTNAM1 DC    CL8'APPXYZSS'  Component Start/Stop
*                                       routine name
          PARM1    DC    CL8'CTAPPXYZ'  PARMLIB member name
          SSRC     DS    F              Return code from Start/Stop
          SSRSN    DS    F              Reason code from Start/Stop
          TCRC     DS    F              Return code from CTRACE
          TCRSN    DS    F              Reason code from CTRACE
```

CTRACE - List form

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

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

Syntax	Description
ƀ	One or more blanks must precede CTRACE.
CTRACE	
ƀ	One or more blanks must follow CTRACE.
,PLISTVER=*xplistver*	*xplistver*: Parameter list version 0, 1, or 2
,PLISTVER=IMPLIED_VERSION	**Default:** IMPLIED_VERSION
,PLISTVER=MAX	**Default:** Version that allows all specified parameters
,MF=(L,*cntl)*	*cntl*: Symbol.
,MF=(L,*cntl,attr)*	*attr*: 1- to 60-character input string.
,MF=(L,*cntl*,0D)	**Default:** 0D.

Parameters

The parameters are explained as follows:

,PLISTVER=(*xplistver* | IMPLIED_VERSION)
Is an optional one byte input, decimal value in the range of 0 to 2, that specifies the macro version associated with CTRACE. PLISTVER is the only keyword allowed on the list form of the macro. It determines which parameter list is generated.

Note that MAX can be specified instead of a number, and the parameter list will be the largest size currently supported. This size may grow from release to release (thus possibly affecting the amount of storage needed by your program). If your program can tolerate this, IBM recommends that you always specify MAX when creating the list form parameter list, to ensure that the list form parameter list is always long enough to hold whatever parameters are specified on the execute form. The macro keywords associated with each supported version of the macro are listed below.

Version	Keyword
0	• ASID • BUFFER • DEFINE • DELETE • DELSUBS • FMTTAB • IFNOSUBS • JOBS • MINOPS • NAME • RC • STARTNAM

Version	Keyword
1	• BUFDEFIN • BUFDFLT • BUFMAX • BUFMIN • HEAD • HEADOPTS • LIKEHEAD • MANYSUB • MOD • PARM • SSRC • SSRSNC • SUB • USERDATA • WTR • WTRMODE
2	• DISPNAM

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

cntl is the name of a storage area for the parameter list.

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

CTRACE - Execute form

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

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

Syntax	Description
ƀ	One or more blanks must follow CTRACE.
DEFINE	
DELETE	
,NAME=*name*	*name*: RX-type address or register (2) - (12).
,STARTNAM=*sname*	*sname*: RX-type address or register (2)-(12).
,DISPNAM=*dname*	*dname*: RX-type address or register (2)-(12).
,PARM=*parm*	*parm*: RX-type address or register (2) - (12).
,PARM=NOPARM	**Default:** PARM=NOPARM.
,ASIDS=NO	**Default:** ASIDS=NO.
,ASIDS=YES	
,JOBS=NO	**Default:** JOBS=NO.
,JOBS=YES	
,MINOPS=*options*	*options*: RX-type address.
,MINOPS=NONE	**Default**: MINOPS=NONE.
,ON=MIN	**Default:**ON=MIN
,ON=NOTMIN	
,MOD=NO	**Default:** MOD=NO.
,MOD=YES	
,FMTTAB=*fmtabs*	*fmtabs*: RX-type address or register (2) - (12).
,FMTTAB=NONE	**Default**: FMTTAB=NONE.
,USERDATA=*userdata*	*userdata*: RX-type address or register (2) - (12).
,USERDATA=NOUSERDATA	**Default**: USERDATA=NOUSERDATA

Syntax	Description
,HEAD=NO	**Default:** HEAD=NO.
,HEAD=YES	
,HEADOPTS=NO	**Default:** HEADOPTS=NO.
,HEADOPTS=YES	
,SUB=*subname*	*subname*: RX-type address or register (2) - (12).
,SUB=NOSUB	**Default:** SUB=NOSUB.
,LIKEHEAD=NO	**Default:** LIKEHEAD=NO.
,LIKEHEAD=YES	
,MANYSUBS=NO	**Default:** MANYSUBS=NO.
,MANYSUBS=YES	
,DELSUBS	**Default**: DELSUBS
,IFNOSUBS	
,BUFFER=NO	**Default:** BUFFER=NO.
,BUFFER=YES	
,BUFDEFIN=NO	**Default:** BUFDEFIN=NO.
,BUFDEFIN=YES	
,BUFMIN=*minsize*	*minsize*: Minimum buffer size.
	Default: BUFMIN=1024.
,BUFMAX=*maxsize*	*maxsize*: Maximum buffer size.
	Default: BUFMAX=2147483647.
,BUFDFLT=*dfltsize*	*dfltsize*: RX-type address or register (2) - (12).
,BUFDFLT=NODFLT	**Default:** BUFDFLT=NODFLT.
,WTR=NO	**Default:** WTR=NO.

Syntax	Description
,WTR=YES	
,WTRMODE=PAGEABLE	
,WTRMODE=DREF	
,WTRMODE=FIXED	
,SSRC=*ss_retcode*	*ss_retcode*: RX-type address or register (2) - (12).
,SSRSNC=*ss_rsncode*	*ss_rsncode*: RX-type address or register (2) - (12).
,RC=*retcode*	*retcode*: RX-type address or register (2) - (12).
,RSNCODE=*rsncode*	*rsncode*: RX-type address or register (2) - (12).
,COM=*comment*	*comment*: A comment string.
,COM=NULL	**Default:** NULL.
,MF=(E,*cntl)*	*cntl*: RX-type address or register (2) - (12).
,MF=(E,*cntl*,COMPLETE)	**Default:** COMPLETE

Parameters

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

,MF=(E,*cntl*)
,MF=(E,*cntl*,COMPLETE)
Specifies the execute form of the macro.

cntl is the name of a storage area for the parameter list.

COMPLETE specifies that the system is to check the macro parameter syntax and supply defaults on parameters that you do not use. COMPLETE is the default.

Chapter 41. CTRACECS — Setting fields in the trace buffer writer control area

Description

The CTRACECS macro allows you to set fields in the trace buffer writer control area (TBWC). By setting these fields, your application can manage and track the status of its trace buffers. When a buffer is full, the application uses the CTRACEWR macro to have the component trace external writer write the buffer out to DASD or tape.

See TBWC in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete field names and lengths, offsets, and descriptions of the fields of the TBWC, which is mapped by the ITTTBWC mapping macro.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24- or 31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled or disabled for I/O and external interrupts
Locks:	No requirement
Control parameters:	Must be in the primary address space or be in an address/data space that is addressable through a public entry on the caller's dispatchable unit access list (DU-AL)

Programming requirements

The program checking the bits in the TBWC must include the ITTTBWC mapping macro.

Restrictions

None.

Register information

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

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14-15
Used as work registers by the system

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One of more blanks must precede CTRACECS.
CTRACECS	
ƀ	One or more blanks must follow CTRACECS.
TBWC=*tbwcaddr*	*tbwcaddr*: RS-type address or register (2) - (12).
,MODE=AVAIL	
,MODE=FULL	
,MODE=FILLING,BUFFSEQ#=seq#addr	
,BUFFSEQ#=*seq#addr*	*seq#addr*: RS-type name or address in register (2) - (12).
,TESTMODE=CURRENT	**Default:** TESTMODE=CURRENT
,TESTMODE=AVAIL	
,TESTMODE=FULL	

Syntax	Description
,TESTMODE=FILLING	
,TESTSEQ#=*testseq#*	*testseq#*: RS-type name or address in register (2)-(12). that specifies the expected buffer sequence number.
,TESTSEQ#=TBWCSEQ#	**Default:** TESTSEQ#=TBWCSEQ#
,CSLABEL=*cslabel*	*cslabel*: A-type name of a label.
	Optional for TESTMODE=CURRENT.
	Required for TESTMODE=AVAIL/ FULL/FILLING and TESTSEQ#.
,CSLABEL=RETRY	**Default:** CSLABEL=RETRY
,COM=*comment*	*comment*: A comment string.
,COM=NULL	**Default:** COM=NULL

Parameters

The parameters are explained as follows:

TBWC=*tbwcaddr*
Specifies the storage for the TBWC which contains the state of the buffer and the buffer sequence number. The storage must be 8 bytes in length and aligned on a doubleword boundary.

If a register is used to give the address of the TBWC and the program is running in access register ASC mode, then the corresponding AR must be set appropriately to contain the ALET of the TBWC.

,MODE=AVAIL
,MODE=FULL
,MODE=FILLING
Indicates the requested state to which the buffer is to be set.

AVAIL
Requests that the trace buffer be set to the available state. Use MODE=AVAIL to initialize the trace buffers to the available state before filling them with data. If the buffer is eventually written out to an external writer data set using the CTRACEWR macro, CTRACE will mark the buffer available when it is finished with it. If the buffer is not going to be written using CTRACEWR, use CTRACECS to mark the buffer available before reusing it.

FILLING
Requests that the buffer be set to the filling state. Use this parameter before you have put any trace entries in the buffer to indicate that it is about to be filled.

IBM recommends that TESTMODE=AVAIL be used with MODE=FILLING to make sure that you will not overlay data in a buffer that is already in use.

FULL
Requests that the buffer be set to the full state. Use this parameter to indicate that the buffer is filled with trace data. No more data should be put into the buffer until its state is set to available. If you are using CTRACEWR, CTRACE will mark the buffer available when it is finished writing its

contents to the output dataset. If you are not using CTRACEWR, you will have to mark the buffer available using the CTRACECS macro, specifying MODE=AVAIL.

,BUFFSEQ#=*seq#addr*
Specifies the name (RS-type) or address (in register 2-12) of a fullword that contains the address of TBWCxxxx, a field that contains the buffer sequence number. The number, starting at one and incremented by 1 for every buffer, must be unique for every buffer passed to an external writer by a given trace.

For MODE=FILLING, the BUFFSEQ# parameter is required. Do not specify BUFFSEQ# with MODE=FULL or AVAIL.

,TESTMODE=CURRENT
,TESTMODE=AVAIL
,TESTMODE=FILLING
,TESTMODE=FULL
Optional input specifying the expected state of the buffer. The expected state is compared to the current state of the buffer. The TBWC is only updated with the requested state (MODE) if the expected state (TESTMODE) is the same as the current state of the buffer.

CURRENT
CURRENT is the default. It sets the state of the buffer to the state specified by the MODE keyword regardless of the current state.

AVAIL
Requests that the state of the buffer be set to the state requested by the MODE keyword only when the buffer is in the available state. Use this parameter with MODE=FILLING to change the state of the buffer to its next valid state. CSLABEL is required with TESTMODE=AVAIL.

FILLING
Requests that the state of the buffer be set to the state requested by the MODE keyword only when the buffer is in the filling state. Use this parameter with MODE=FULL to change the state of the buffer to its next valid state. CSLABEL is required with TESTMODE=FILLING.

FULL
Requests that the state of the buffer be set to the state requested by the MODE keyword only when the buffer is in the full state. Use this parameter with MODE=AVAIL to change the state of the buffer to its next valid state. CSLABEL is required with TESTMODE=FULL.

,TESTSEQ#=*testseq#*
,TESTSEQ#=TBWCSEQ#
Optional fullword input value that is used to test the current buffer sequence number. If the input value matches the current buffer sequence number the TBWC is updated to the expected requested state (specified by the MODE keyword). If you are using more than one buffer, TESTSEQ# ensures that your are changing the state of the correct buffer by verifying its sequence number in the TBWC.

,CSLABEL=*label*
,CSLABEL=RETRY
Specifies the name of a label within your application to which the system returns control when the current mode or sequence number does not equal the expected buffer mode or sequence number. CSLABEL is required when using the TESTMODE and TESTSEQ# keywords; however, it is optional when used with TESTMODE=CURRENT.

If CSLABEL=RETRY is specified, the application will branch to a system generated label that retries the CDS instruction with the current value of the TBWC. CSLABEL=RETRY is not valid with the TESTMODE and TESTSEQ# parameters. It is provided for existing applications that invoked the CTRACECS macro before the two parameters became available.

,COM=*comment*
,COM=NULL
Optional input. Comments the macro invocation. The comment string must be enclosed in quotation marks if it contains any lowercase characters.

Return and reason codes

None.

Example 1

Indicate to component trace that you are starting to fill a trace buffer. Then indicate to component trace that the buffer is full. Note that this example does not use the, TESTMODE TESTSEQ#, and CSLABEL parameters which would prevent buffers from being overwritten, especially in a sysplex environment.

```
         CTRACECS TBWC=TBWCAREA,MODE=FILLING,BUFFSEQ#=REQ#
             .
             .
             .
         CTRACECS TBWC=TBWCAREA,MODE=FULL

         TBWCAREA DS CL8               Trace Buffer Writer Control area
*                                      TBWC contains buffer seq #
         REQ#     DS F                 Buffer Seq #
*
```

Example 2

Test to ensure that the next buffer associated with the input TBWC is currently available. If it is, update the mode to FILLING. If the buffer is not available the code will branch to the subroutine TLABL.

```
           CTRACECS TBWC=TBWCAREA,MODE=FILLING,BUFFSEQ#=next#,
                  TESTMODE=AVAIL,CSLABEL=TLABL

             .

             .

           TLABL: (next instruction)
```

Example 3

Update the status of the buffer from FILLING to FULL if its sequence number is equal to the expected buffer sequence number in the TESTSEQ# parameter.

```
         CTRACECS TBWC=TBWCAREA,MODE=FULL,TESTMODE=FILLING,
                  TESTSEQ#=xtestseq#,CSLABEL=TLABL
```

Example 4

Update the status of the buffer to FILLING regardless of its current status. The following example uses the default values for TESTMODE and CSLABEL.

```
         CTRACECS TBWC=TBWCPTR,MODE=FILLING,BUFFSEQ#=BUFFNUM,
                  TESTMODE=CURRENT,CSLABEL=RETRY
```

Chapter 42. CTRACEWR — Write a full trace buffer to DASD or tape

Description

The CTRACEWR macro enables the component trace external writer to write a full trace buffer out to a trace data set on DASD or tape.

The CTRACEWR macro will asynchronously capture a full trace buffer while the application continues processing and writing trace entries to another trace buffer.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or PSW key 0-7
Dispatchable unit mode:	Task or SRB mode
Cross memory mode:	PASN=HASN=SASN or PASN¬=HASN¬=SASN
AMODE:	31-bit.
ASC mode:	Primary or access register.
Interrupt status:	Enabled or disabled for I/O and external interrupts.
Locks:	If SYNCH(YES) is specified, no locks can be held.
Control Parameters:	Must be in the primary address space

Programming requirements

None.

Restrictions

If either the BUFFALET or the TBWCALET identifies the secondary or home address space, then both must identify the same address space (that is, both the trace buffer and the trace buffer writer control area must be in the same address space).

Register information

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

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

Register
Contents

0
If GPR 15 contains 0 or 4, GPR 0 is used as a work register by the system; otherwise, GPR 0 contains a reason code.

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the ARs contain:

Register
Contents

0-15
Unchanged

Performance implications

None.

Syntax

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CTRACEWR.
CTRACEWR	
ƀ	One or more blanks must follow CTRACEWR.
BUFFADDR=*buffer_address*	*buffer_address*: RS-type address or register (2)-(12).
,BUFFALET=*buffer_alet*	*buffer_alet*: RS-type address or register (2)-(12).
,BUFFALET=NOBUFFALET	**Default:** BUFFALET=NOBUFFALET
,BUFFLEN=*buffer_length*	*buffer_length*: RS-type address or register (2)-(12).
,TOKEN=*token*	*token*: RS-type address or register (2)-(12).
,TBWCADDR=*tbwc_address*	*tbwc_address*: RS-type address or register (2)-(12).

Syntax	Description
,TBWCALET=*tbwc_alet*	*tbwc_alet*: RS-type address or register (2)-(12).
,TBWCALET=NOTBWCALET	**Default:** TBWCALET=NOTBWCALET
,SYNCH=YES | NO	**Default:** SYNCH=NO
,RC=*return_code*	*return_code*: RS-type address or register (2)-(12).
,RSNCODE=*reason_code*	*reason_code*: RS-type address or register (2)-(12).
,COM=*comment*	*comment*: A comment string.
,COM=NULL	**Default:** COM=NULL.
,MF=(S)	**Default:** MF=(S)

Parameters

The parameters are explained as follows:

BUFFADDR=*buffer_address*
Specifies a required parameter that points to the address of the buffer to be written externally.

,BUFFALET=*buffer_alet*
,BUFFALET=NOBUFFALET
Contains the PASN ALET that identifies the address/data space where the buffer resides. Use this optional parameter when the buffer to be written externally resides in either a data space or an address space that is different from the current primary address space. The default is BUFFALET=NOBUFFALET.

,BUFFLEN=*buffer_length*
A required parameter that indicates the number of bytes in length of the buffer to be written externally. IBM recommends the length be at least 4KB. Component trace will split buffers that are too large to fit into a single block.

,TOKEN=*token*
A required parameter that specifies the token passed to the start/stop exit routine when it was requested to start tracing externally.

TBWCADDR=*tbwc_address*
Specifies a required parameter that points to a word that points to the address of the storage obtained by the application for the trace buffer writer control area (TBWC) mapped by ITTTBWC. The TBWC provides communication between the application and component trace. See TBWC in *z/OS MVS Data Areas* in the z/OS Internet library (www.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosInternetLibrary) for complete field names and lengths, offsets, and descriptions of the fields of the TBWC.

,TBWCALET=*tbwc_alet*
,TBWCALET=NOTBWCALET
Contains the ALET that identifies the address/data space where the TBWC resides. Use this optional parameter when the TBWC resides in either a data space or an address space that is different from the current primary address space. The default is TBWCALET=NOTBWCALET.

,SYNCH=YES | NO
YES causes CTRACE to copy the application's buffers before control is returned instead of scheduling an asynchronous SRB to copy the buffer. The CTRACEWR function executes synchronously. The SYNCH keyword is optional. NO causes the CTRACEWR function to execute asynchronously.

Note: Because your application will run slower, IBM does not recommend that you use the SYNCH keyword on every CTRACEWR invocation. Use the SYNCH keyword in the start/stop routine any time that the trace buffers will be freed. For example, when the trace is being turned off or the buffer size is changing, you can free trace buffer storage after issuing the CTRACEWR macro with the SYNCH keyword and be assured that the buffers were copied to I/O buffers to be written to the external data set by CTRACE. The default is SYNCH=NO.

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

,RSNCODE=*reason_code*
Specifies the location where the system is to store the reason code. If GPR 15 contains a return code other than 0 or 4, the reason code is also in GPR 0.

,COM=*comment*
,COM=NULL
Comments the macro invocation. If the comment contains any lowercase characters, it must be enclosed in quotation marks.

,MF=(S)
Specifies the standard form of the CTRACEWR macro.

ABEND codes

The following table identifies abend code and reason code combinations, and a description of what each means:

Table 51. Abend codes for the CTRACEWR Macro

Abend Code	Reason Code	Description
00D	00010100	For the CTRACEWR macro, the parameter list version number is not correct.
00D	00010200	The system found either nonzero values in the reserved fields or unused fields for the requested service in the CTRACEWR macro parameter list.
00D	00010300	The buffer length passed was 0 or less.

Return and reason codes

When control returns from CTRACEWR, GPR 15 (and *return_code*, if you coded RC) contains one of the following return codes. The third byte of GPR 0 (and *reason_code*, if you coded RSNCODE) might contain one of the following reason codes.

Note: An application should always check the return code from the CTRACEWR macro. A non-zero code indicates that some data might have been lost in the next record output.

Table 52. Return and Reason Codes for the CTRACEWR Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning
00	None.	CTRACEWR was successful.
04	None.	CTRACEWR was unsuccessful. No data was captured because the trace is not connected to an active external writer.
08	xxxx01xx	Storage required to perform the write operation could not be obtained.
08	xxxx02xx	CTRACEWR was unable to schedule an SRB to process this request.
08	xxxx03xx	The control information (TBWC) has already been reused by the application.

Table 52. Return and Reason Codes for the CTRACEWR Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning**
0C	xxxx01xx	The caller is holding locks.
0C	xxxx02xx	The input token was not valid.
0C	xxxx0300	The TBWC is not valid because the sequence number is the same as a previous write request.
0C	xxxx0301	The TBWC is not valid for one of the following reasons: • The TBWC is not in central storage and the CTRACEWR issuer is disabled. • The BUFFALET is not the same as the TBWCALET.

Example

Indicate to component trace that the buffer at address TRACEADR is ready to be written out. Pass the token (TCWTRTKN) that the application received from the start/stop routine. Component trace is to store the return and reason codes from the CTRACEWR macro in TCRCODE and TCRSNCODE.

```
         CTRACEWR BUFFADDR=TRACEADR,BUFFLEN=TRACESIZ,                  X
               TOKEN=TCWTRTKN,TBWCADDR=TBWCADR,                        X
               RC=TCRCODE,RSNCODE=TCRSNCODE

         TBWCADR   DS A               TBWC address
         TRACEADR  DS A               Trace buffer address
         TRACESIZ  DS F               Trace buffer size
         TCWTRTKN  DS CL8             Trace writer token produced by
*                                     CTRACE upon connection
         TCRCODE   DS F               Return code from CTRACE
         TCRSNCODE DS F               Reason code from CTRACE
```

CTRACEWR - List form

Syntax

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

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

Parameters

The parameters are explained as follows:

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

cntl is the name of a storage area for the parameter list.

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

CTRACEWR - Execute form

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

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede CTRACEWR.
CTRACEWR	
ƀ	One or more blanks must follow CTRACEWR.
BUFFADDR=*buffer_address*	*buffer_address*: RS-type address or register (2)-(12).
,BUFFALET=*buffer_alet*	*buffer_alet*: RS-type address or register (2)-(12).
,BUFFALET=NOBUFFALET	**Default:** BUFFALET=NOBUFFALET
,BUFFLEN=*buffer_length*	*buffer_length*: RS-type address or register (2)-(12).
,TOKEN=*token*	*token*: RS-type address or register (2)-(12).
,TBWCADDR=*tbwc_address*	*tbwc_address*: RS-type address or register (2)-(12).

Syntax	Description
,TBWCALET=*tbwc_alet*	*tbwc_alet*: RS-type address or register (2)-(12).
,TBWCALET=NOTBWCALET	**Default:** TBWCALET=NOTBWCALET
,SYNCH=YES | NO	**Default:** SYNCH=NO
,RC=*return_code*	*return_code*: RS-type address or register (2)-(12).
,RSNCODE=*reason_code*	*reason_code*: RS-type address or register (2)-(12).
,COM=*comment*	*comment*: A comment string.
,COM=NULL	**Default:** COM=NULL.
,MF=(E,*cntl*)	*cntl*: RX-type address or register (2) - (12).
,MF=(E,*cntl*,COMPLETE)	**Default:** COMPLETE

Parameters

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

,MF=(E,*cntl*)
,MF=(E,*cntl*,COMPLETE)
Specifies the execute form of the macro.

cntl is the name of a storage area for the parameter list.

COMPLETE specifies that the system is to check the macro parameter syntax and supply defaults on parameters that you do not use. COMPLETE is the default.

Chapter 43. DATOFF — DAT-OFF linkage

Description

The DATOFF macro transfers control to a specified routine in the DAT-OFF section of the nucleus.

The macro is restricted to key 0, supervisor state users, that are enabled for DAT. Callers must include the IHAPSA mapping macro with the DATOFF macro. Callers can be in primary or access register (AR) address space control (ASC) mode. The macro destroys the contents of general registers 0, 14, and 15.

Environment

These are the requirements for the caller:

Environmental factor	Requirement
Minimum authorization:	Supervisor state and key 0
Dispatchable unit mode:	Task or SRB
Cross memory mode:	PASN=HASN=SASN
AMODE:	24-, 31- or 64-bit
ASC mode:	Primary or Access Register
Interrupt status:	None
Locks:	The caller may hold locks, but is not required to hold any.
Control parameters:	Must be in the primary address space

Programming requirements

The caller must include the IHAPSA mapping macro and must be enabled for DAT.

Restrictions

None.

Input register information

Before issuing the DATOFF 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

2-5
May be used as work registers depending upon the index specified

6-13
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

Performance implications

None.

Syntax

The DATOFF macro is written as follows:

Syntax	Description
name	*name*: Symbol. Begin *name* in column 1.
ƀ	One or more blanks must precede DATOFF.
DATOFF	
ƀ	One or more blanks must follow DATOFF.
index	**Note:** See the description of the parameters for the valid options.
,RELATED=*value*	*value*: Any valid macro keyword specification.

Parameters

The parameters are explained as follows:

index
Specifies the function that is to be given control in the DAT-OFF section of the nucleus. The possible values for *index*, along with the associated functions, are as follows:

Index
Function

INDCDS
31-bit DAT-OFF compare double and swap

INDMVCL0
31-bit general DAT-OFF move character long

INDMVCLK
31-bit general DAT-OFF move character long in user key

INDXC0
31-bit general DAT-OFF exclusive OR character

INDCDS64
64-bit DAT-OFF compare double and swap

INDMVCL64
64-bit general DAT-OFF move character long

INDMVCLK64
64-bit general DAT-OFF move character long in user key

INDXC64
64-bit general DAT-OFF exclusive OR character

INDUSR1
User-written 31-bit

INDUSR2
User-written 31-bit

INDUSR3
User-written 31-bit

INDUSR4
User-written 31-bit

INDUSR641
User-written 64-bit

INDUSR642
User-written 64-bit

INDUSR643
User-written 64-bit

INDUSR644
User-written 64-bit

For all system-defined index values (INDCDS, INDMVCL0, INDMVCLK, and INDXC0), the user must supply information in certain registers, as shown in the following lists. All register values for INDCDS, INDMVCL0, INDMVCLK, and INDXC0 must be 31-bit addresses. All register values for INDCDS64, INDMVCL64, INDMVCLK64, and INDXC64 must be 64-bit addresses. Callers must be AMODE=64 to use INDCDS64, INDMVCL64, INDMVCLK64, or INDXC64.

INDCDS and INDCDS64

Registers
Information

2-3
First 64-bit operand in even-odd pair of registers (target data)

4-5
Third 64-bit operand in even-odd pair of registers (source data)

6
Location of second operand, a doubleword in storage (target address)

Note: Register 6 contains a real address. If INCDS is specified, then the low order 32 bits of GPR 6 form the real address operand. If INCDS64 is specified, then all 64 bits of GPR6 form the real address operand.

INDMVCL0 and INDMVCL64

Registers	Information
2	Location into which the characters are to be moved
3	Length of the area into which the characters are to be moved
4	Location of the area from which the characters are to be moved
5	Length of the area from which the characters are to be moved

Note: Registers 2 and 4 contain real addresses.

INDMVCLK and INDMVCLK64

Registers	Information
2	Location into which the characters are to be moved
3	Length of the area into which the characters are to be moved
4	Location of the area from which the characters are to be moved
5	Length of the area from which the characters are to be moved
6	Bits 24-27 contain the PSW key in which the MVCL is to be executed

Note: Registers 2 and 4 contain real addresses.

INDXC0 and INDXC64

Registers	Information
2	Location of the results of exclusive OR character processing
3	Bits 24-31 contain one less than the number of bytes on which the exclusive OR is to be performed
4	Location of the operand on which the exclusive OR is to be performed

Note: Registers 2 and 4 contain real addresses.

There are eight DAT-OFF indexes that users can define. These indexes are INDUSR1, INDUSR2, INDUSR3, INDUSR4, INDUSR641, INDUSR642, INDUSR643, and INDUSR644. User written DAT-OFF functions are restricted as follows:

- The user of the DATOFF macro instruction must be in key 0, supervisor state, and executing with DAT turned on.
- The DAT-OFF function must have the attributes AMODE=31 and RMODE=ANY to use INDUSR1, INDUSR2, INDUSR3, and INDUSR4.
- The DAT-OFF function must have the attributes AMODE=64 and RMODE=ANY to use INDUSR641, INDUSR642, INDUSR643, and INDUSR644.
- The DAT-OFF function must preserve register 0 because register 0 contains the return address of the module that issued the DATOFF macro.

- The DAT-OFF function must use branch instructions to link to other DAT-OFF functions.
- The DAT-OFF function must use BSM 0,14 to return from INDUSR1, INDUSR2, INDUSR3, and INDUSR4.
- The DAT-OFF function must return via BR 14 from INDUSR641, INDUSR642, INDUSR643, and INDUSR644.

Note: See *z/OS MVS Programming: Authorized Assembler Services Guide* for information about how to insert a user-written function in the nucleus.

,RELATED=*value*
Specifies information used to document the macro and to relate the service performed to some corresponding service or function. The format of the information specified can be any valid coding values that the user chooses.

ABEND codes

0FF

See *z/OS MVS System Codes* for an explanation and programmer responses for these codes.

Return codes

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

Table 53. Return Codes for the DATOFF Macro

Hexadecimal Return Code	Meaning and Action
00	**Meaning**: Successful completion. **Action**: None.
04	**Meaning**: The first and second operands specified on INDCDS were not equal (a condition code 1 on the CDS). The first operand has been replaced by the second. **Action**: Reissue the request.

Examples

See *z/OS MVS Programming: Authorized Assembler Services Guide* for examples.

Chapter 44. 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.

A description of the DEQ macro also appears in *z/OS MVS Programming: Assembler Services Reference ABE-HSP* with the exception of the RMC, GENERIC, TCB, and UCB parameters. See the *z/OS MVS Programming: Authorized Assembler Services Guide* for information on using DEQ to release serialization of a resource.

Environment

The requirements for callers of DEQ are:

Environmental factor	Requirement
Minimum authorization:	Problem state with any PSW key. For the RMC, TCB, GENERIC=YES, and UCB (where UCB is not allocated to the requesting task) or when the specified *qname* is ADRDFRAG, ADRDSN, ARCENQG, BWODSN, SYSZ*, SYSCTLG, SYSDSN, SYSIEA01, SYSIEECT, SYSIEFSD, SYSIGGV1, SYSIGGV2, SYSPSWRD, SYSVSAM, or SYSVTOC. Authorization must be **one of the following**: • Supervisor state • PSW key 0-7 • APF-authorized.
Dispatchable unit mode:	Task
Cross memory mode:	For LINKAGE=SVC: PASN=HASN=SASN For LINKAGE=SYSTEM: 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. With the exception of TCB and UCB, all parameters can reside above 16 megabytes.

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.
(	
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	
,RMC=NONE	**Default:** RMC=NONE
,RMC=STEP	
,GENERIC=NO	**Default:** GENERIC=NO
,GENERIC=YES	**Note:** If GENERIC=YES is specified, you must also specify RET=HAVE.
,TCB=*tcb addr*	*tcb addr*: A-type address, or register (2) - (12).

Syntax	Description
	Note: Do not specify TCB with RMC.
,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, RMC, TCB, 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 or if the ECB parameter was specified on the associated ENQ macro. 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.

In either case, if the resources requested for release were originally queued with the ECB parameter specified, they are released with return code 0.

,RMC=NONE
,RMC=STEP
RMC specifies that the reset must-complete function is not to be used (NONE) or that the requesting task is to release the resources and end the must-complete function (STEP). Do not specify RMC with TCB or GENERIC. The NONE or STEP subparameter must agree with the subparameter specified in the SMC parameter of the corresponding ENQ macro. RMC=NONE is the default.

In either case, if the resources requested for release were originally queued with the ECB parameter specified, they are released with return code 0.

,GENERIC=NO
,GENERIC=YES
Specifies whether or not (YES or NO) all resources with the specified *qname* are to be released. For the resource to be released, the task either must have control of the resource, or must be waiting for the system to post the ECB specified on the associated ENQ macro. If the task is waiting for a resource, but is not waiting for the ECB to be posted, the task remains queued and waiting. GENERIC=NO is the default.

,TCB=*tcb addr*
Specifies a register that points to a TCB or specifies the address of a fullword on a fullword boundary that points to a TCB on whose behalf the DEQ is to be done. The caller (not the directed task) ends abnormally if the RET parameter is omitted and an attempt is made to release a resource not requested or not owned by the directed task, except when ECB was specified on the original ENQ. If ECB was specified on the ENQ and the resource is not owned by the directed task, the DEQ request releases the resources with a return code of 0

Note: The TCB resides in storage below 16 megabytes in the caller's home address space.

,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.

- If TCB= is specified, then the specified TCB in the home address space is associated with the resource; otherwise, the TCB in the home address space making the request is associated with the resource.

The default is LINKAGE=SVC.

ABEND codes

For only unconditional requests, the caller might encounter abend code X'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.

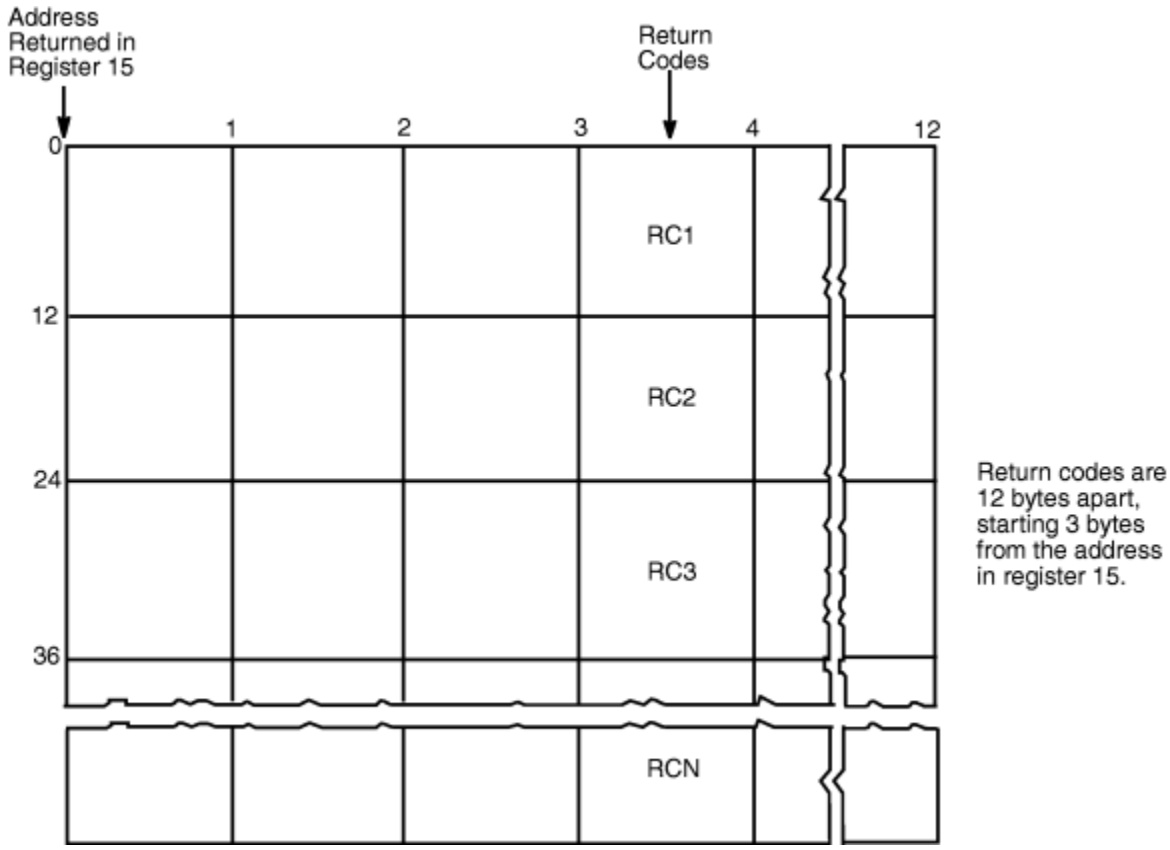

Figure 4. Return Code Area Used by DEQ

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 54. 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 (or resources, if you specified GENERIC=YES). **Action**: None.
4	**Meaning**: The resource (or resources, if you specified GENERIC=YES) 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 (or resources, if you specified GENERIC=YES) 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

Unconditionally release control of the resource in Example 1 of ENQ (see *z/OS MVS Programming: Authorized Assembler Services Reference EDT-IXG*), and reset the “must-complete” state.

```
DEQ (MAJOR1,MINOR1,8,STEP),RMC=STEP
```

Example 2

Conditionally release control of the resource in Example 2 of ENQ.

```
DEQ (MAJOR2,MINOR2,4,SYSTEM),TCB=(R2),RET=HAVE
```

Example 3

Unconditionally release control of the resource (device) in Example 1 of RESERVE (see *z/OS MVS Programming: Authorized Assembler Services Reference LLA-SDU*).

```
DEQ (MAJOR3,MINOR3,,SYSTEMS),UCB=(R3)
```

Example 4

Release control of the resource in Example 1 of ENQ, if it has been assigned to the current TCB. The length of the rname is explicitly defined as 8 characters.

```
DEQ (MAJOR1,MINOR1,8,STEP),RET=HAVE
```

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	

Syntax	Description
,	**Default**: STEP
,STEP	
,SYSTEM	
,SYSTEMS	
)	
,RET=NONE	**Default**: RET=NONE
,RET=HAVE	
,RMC=NONE	**Default**: RMC=NONE
,RMC=STEP	
,GENERIC=NO	**Default**: GENERIC=NO
,GENERIC=YES	**Note**: If GENERIC=YES is specified, you must also specify RET=HAVE.
,TCB=0	**Note**: TCB cannot be specified with RMC, and must be specified on the list form if used on the execute form.
,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.
(	**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	

Syntax	Description
,RET=HAVE	
,RMC=NONE	
,RMC=STEP	
,GENERIC=NO	
,GENERIC=YES	**Note:** If GENERIC=YES is specified, you must also specify RET=HAVE.
,TCB=*tcb addr*	*tcb addr*: RX-type address, or register (2) - (12).
	Note: TCB cannot be specified with RMC and must be specified on the execute form if used on the list form.
,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	
,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 45. DIV — Data-in-virtual

Description

The DIV macro establishes a window in your address space, data space, or hiperspace and enables your program to reference or 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, you use assembler instructions to access data. If the window is in a hiperspace, you 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.

For guidance information on the use of data-in-virtual, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	To use the TTOKEN or CHECKING parameters, programs must be in supervisor state with PSW key 0-7. Programs that use other parameters can be in problem state with any PSW key. Additionally, a program requesting a data-in-virtual service under a given ID must be running with PSW key 0 or the same key as the program that obtained the ID.

Environmental factor	Requirement
Dispatchable unit mode:	Task
Cross memory mode:	PASN=HASN=SASN
AMODE:	31-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts
Locks:	No locks held
Control parameters:	Must be in the primary address space 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

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

- The task that obtains the ID is the only task that can issue DIV ACCESS against that ID. Any authorized (supervisor state, PSW key 0 - 7, or APF-authorized) subtask of the obtaining task can issue a DIV service that specifies the given ID, with one exception: if the object is a hiperspace, then the service cannot be MAP or SAVE.
- 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 you are in cross memory mode, you cannot invoke data-in-virtual services; however, you can reference and update data in a mapped virtual storage window.
- Tasks that are unauthorized cannot issue DIV services with an ID that belongs to another task.
- When you identify a data-in-virtual object using the IDENTIFY service, you cannot request a checkpoint until you invoke the corresponding UNIDENTIFY service.
- DIV does not support VSAM extended format linear data sets for use as a DIV object for which the size is greater than 4GB.
- When you use DIV with the IARVSERV macro to share data in virtual storage, you must meet several requirements. For those requirements, see the chapter about sharing data through IARVSERV in *z/OS MVS Programming: Authorized Assembler Services Guide*.

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

0-1
Used as work registers by the system

2-13
Unchanged

14-5
Used as work registers by the system

Performance implications

- By using the DIV macro, you are likely to 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 which have changed.
- Using LOCVIEW=MAP on a DIV ACCESS request degrades performance. Use LOCVIEW=NONE whenever possible. You can use LOCVIEW=MAP 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.

Syntax	Description
	Valid parameters: (Underlined parameters are those that you must specify.)
IDENTIFY	ID, TYPE, DDNAME *or* STOKEN, CHECKING, TTOKEN
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).
,CHECKING=YES	**Default:** CHECKING=YES
,CHECKING=NO	
,DDNAME=*addr*	*addr*: RX-type address, or register (2) - (12).
,LISTADDR=*listaddr*	*listaddr*: RX-type address, or register (2) - (12).
,LISTSIZE=*listsize*	*listsize*: 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

Syntax	Description
,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.
,TTOKEN=*	**Default:** TTOKEN=*
,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 and their keywords 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. 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. To bypass data-in-virtual validity checking, code CHECKING=NO. To assign ownership of the ID to another task, code TTOKEN=*addr*.

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 disabled reference (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 disabled reference (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 either be 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 with 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.

,CHECKING=YES
,CHECKING=NO
To have data-in-virtual perform validity checking, code CHECKING=YES, or omit the CHECKING parameter; CHECKING=YES is the default.

To bypass data-in-virtual validity checking for the corresponding ID, code CHECKING=NO. The calling program must ensure the validity of the parameter list, the parameter values, and the environment at the time the DIV macro is issued. If a parameter or the environment is not valid, the results are unpredictable. Data-in-virtual also bypasses validity checking for other invocations of the DIV macro that use the same ID. Bypass data-in-virtual validity checking only if you need to improve data-in-virtual performance.

,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=*addr*
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=*addr*
Specifies the address of a 4-byte field that contains the number of entries in the user list for 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 or default to LOCVIEW=NONE.

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 or default to RETAIN=NO 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 by the MAP, RESET, or SAVE services. These services 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 (0), 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.

Specify SPAN only when you specify MAP, RESET, or SAVE.

,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, 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.

,TTOKEN=*addr*
,TTOKEN=*
Assigns ownership of the corresponding ID to a TCB.

To assign ownership to your TCB, code TTOKEN=*, or omit TTOKEN.

To assign ownership to another TCB, code TTOKEN=*addr*. *addr* identifies a 16-byte location that contains the TTOKEN of the task to be assigned ownership. You can assign ownership to another TCB only when the other TCB and your TCB are in the same chain, and the other TCB is higher in the chain than your TCB.

,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 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 DIV 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 real storage.

,RELEASE=YES
,RELEASE=NO
Specify RELEASE=YES to release all virtual pages in the reset range. Specify or default to RELEASE=NO 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

The DIV macro might abnormally terminate with abend code X'08B'. See *z/OS MVS System Codes* for an explanation and programmer responses.

Return and reason codes

When the system returns control to the caller after the caller invokes the DIV macro, 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 ends 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 return and reason codes are shown in the following table.

Table 55. Return and reason codes for the DIV macro

Hexadecimal return code	Hexadecimal reason code	Meaning and action
00	None.	**Meaning**: Successful completion. **Action**: None.
04	001A	**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.
04	002D	**Meaning**: The data space has been deleted. However, the requested operation completed successfully. **Action**: None.
04	0037	**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.
04	0043	**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.
04	0044	**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 55. Return and reason codes for the DIV macro (continued)

Hexadecimal return code	Hexadecimal reason code	Meaning and action
04	0802	**Meaning**: The caller invoked DIV UNIDENTIFY or UNACCESS. The function completed successfully, but with exceptional circumstances. **Action**: None required.
04	0807	**Meaning**: Environmental error. Media damage may 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.
08	000A	**Meaning**: Environmental error. There is another service currently executing with the specified ID. **Action**: Retry the request one or more times until the other service currently executing for this ID completes.
08	001C	**Meaning**: Environmental error. The object cannot be accessed at the current time. **Action**: Retry the request one or more times until the operation succeeds.
08	0028	**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.
08	003E	**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.
08	0040	**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.
08	0045	**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.
08	004E	**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.
08	0801	**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.
08	0802	**Meaning**: System error. I/O driver failure. **Action**: Retry the request. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel.

Table 55. Return and reason codes for the DIV macro (continued)

Hexadecimal return code	Hexadecimal reason code	Meaning and action
08	0808	**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 codes and supply them to the appropriate IBM support personnel.
08	0809	**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.
08	080A	**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.
08	080B	**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.
08	080C	**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.
0C	0017	**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 codes and supply them to the appropriate IBM support personnel.
0C	0021	**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 codes and supply them to the appropriate IBM support personnel.
0C	0803	**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 codes and supply them to the appropriate IBM support personnel.
0C	0804	**Meaning**: System error. Catalog update failed. **Action**: Retry the request. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel.
0C	0806	**Meaning**: System error. I/O error. **Action**: Retry the request. If the problem persists, record the return and reason codes and supply them to the appropriate IBM support personnel.

Example 1

Identify a hiperspace as a data object. The hiperspace's STOKEN is at HSSTOK. IDENTIFY is to return the ID at DIVOBJID.

```
DIV IDENTIFY,TYPE=HS,STOKEN=HSSTOK,ID=DIVOBJID
```

Example 2

Whenever a page fault on a page in the mapped range requires that the system read the page from the data set object, the system, if possible, preloads up to seven additional pages, virtually successive to the fault page.

```
DIV MAP,ID=DIVOBJID,AREA=MAPPTR1,SPAN=SPANVAL,
OFFSET=*,STOKEN=DSSTOK,PFCOUNT=7
```

DIV - List form

Syntax

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

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 , STOKEN, CHECKING, TTOKEN
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*: A-type address
,AREA=*addr*	*addr*: A-type address
,CHECKING=YES	**Default:** CHECKING=YES

Syntax	Description
,CHECKING=NO	
,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
,TTOKEN=*	**Default:** TTOKEN=*
,TYPE=DA	**Default:** None
,TYPE=HS	
,PFCOUNT=*nnn*	**Default:** 0

Syntax	Description
,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 , STOKEN, CHECKING, TTOKEN
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

Syntax	Description
,ID=*addr*	*addr*: RX-type address, or register (2) - (12).
,AREA=*addr*	*addr*: RX-type address, or register (2) - (12).
,CHECKING=YES	**Default:** CHECKING=YES
,CHECKING=NO	
,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.
,TTOKEN=*addr*	*addr*: RX-type address, or register (2) - (12).
,TTOKEN=*	**Default:** TTOKEN=*

Syntax	Description
,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 number. 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 , STOKEN, CHECKING, TTOKEN
ACCESS	ID, MODE, SIZE, LOCVIEW

Syntax	Description
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).
,CHECKING=YES	**Default:** CHECKING=YES
,CHECKING=NO	
,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=*	

Syntax	Description
,SPAN=*addr*	*addr*: RX-type address, or register (2) - (12).
,SPAN=*	
,STOKEN=*addr*	*addr*: RX-type address
,TTOKEN=*addr*	*addr*: RX-type address or register (2) - (12).
,TTOKEN=*	**Default:** TTOKEN=*
,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.

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 the same as the EXECUTE form except that the data-in-virtual service is not called. The contents of registers 1 and 15 are destroyed.

Chapter 46. 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 and from the list of messages retained by MVS. When a program no longer requires that a message be displayed or retained, it issues the DOM macro to delete the message.

When a program issues a WTO or WTOR macro, the system assigns an identification number to the message and returns this number in register 1 to the issuing program. When the display of this message is no longer needed, the issuing program can issue the DOM macro using the identification number that was returned in register 1.

MVS automatically invokes DOM for a WTOR when it receives the WTOR reply. You need only invoke DOM for a WTOR if the WTOR becomes obsolete before the system receives the reply. When the system receives the reply, the message is no longer highlighted on the MCS console screen. The message is eventually removed from the screen of an MCS console.

Environment

The requirements for the caller are different for LINKAGE=SVC and LINKAGE=BRANCH.

If you specify LINKAGE=SVC, the requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state or 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

If you specify LINKAGE=BRANCH, the requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Supervisor state with PSW key 0-7
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	24- or 31-bit
ASC mode:	Primary
Interrupt Status:	Enabled or disabled for I/O and external interrupts
Locks:	The caller may hold locks, but is not required to hold any
Control parameters:	Must be in the primary address space.

Programming requirements

None.

Restrictions

The following restrictions and limitations apply:

- If the caller is executing in a sysplex, do not use SCOPE=SYSTEM, but rather SCOPE=SYSTEMS. DOM ignores SCOPE=SYSTEM if the caller is executing in a sysplex.
- For any DOM parameters that allow a register specification, the value must be right-justified in the register, and the remaining bytes within the register must be zero.
- Any authorized DOM parameters that are specified by an unauthorized program will cause a 157 abend. The only authorized parameter is SCOPE.

Register information

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

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

Register
Contents

0-1
Used as work registers by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code if you specified LINKAGE=BRANCH; otherwise, used as a work register by the system.

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.

Syntax	Description
MSG=*field*	*field*: Four-byte value
MSGLIST=*list addr*	*list addr*: Symbol, RX-type address, or register (1) - (12).
TOKEN=*addr*	*addr*: Register (1) - (12), or an RX-type address.
,COUNT=*count addr*	*count addr*: Register (2) - (12), or an RX-type address.
,SCOPE=SYSTEM	**Default:** SCOPE=SYSTEMS
,SCOPE=SYSTEMS	
,LINKAGE=SVC	**Default:** LINKAGE=SVC
,LINKAGE=BRANCH	

Parameters

The parameters are explained as follows:

MSG=*field*
MSGLIST=*list addr*
TOKEN=*addr*

MSG=*field* specifies in register 1 the ID of a single message to be deleted. Register 1 contains the 32-bit identification number (the DOM id) of a WTO message to be deleted. Use this as input on the MSG parameter.

MSGLIST=*list addr* specifies the address of a list of one or more fullwords, each word containing the 32-bit identification number of a message to be deleted. You can end the list in one of two ways:

- Use the COUNT parameter to specify how many entries are in the list.
- Turn on the high-order bit in the last entry of the list.

Attention: Do not alter a DOM id 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. When you issue WTO or WTOR to write a message, you can choose a token value, and specify it as an input parameter to WTO or WTOR through the TOKEN parameter. To issue a DOM using a TOKEN, ignore the message ID returned by WTO or WTOR in register 1, and specify the token value instead, using the TOKEN parameter when you issue DOM. TOKEN is an alternate method for identifying messages, which is independent of the register 1 message ID. Specifying TOKEN with DOM will delete all messages specified with WTO and that particular TOKEN.

With TOKEN, authorized users may delete any messages originally issued under the same ASID and system ID. Unauthorized users may delete only those messages that 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. TOKEN is mutually exclusive with MSG, MSGLIST, and COUNT.

,COUNT=*count addr*
Specifies a field or register containing the 1-byte count of 4-byte message IDs associated with this request. The count must be from 1 to 60. If you specify COUNT, do not set the high-order bit on in the last entry of the DOM parameter list. If you specify COUNT in a register, ensure that it is right-justified and padded with zeros. If you do not specify COUNT, the message IDs are treated as 32-bit IDs. If an address is used, the address points to a 1-byte field that contains the count. COUNT is not valid with TOKEN.

,SCOPE=SYSTEM
,SCOPE=SYSTEMS
Specifies how to process the DOM request. If you specify SCOPE=SYSTEMS, the DOM request is to be communicated to other processors. If you specify SCOPE=SYSTEM, the DOM request is not to be communicated to other processors. If you do not specify SCOPE, the DOM request defaults to SCOPE=SYSTEMS.

You must specify SCOPE=SYSTEMS if you are running in a sysplex.

,LINKAGE=SVC
,LINKAGE=BRANCH
Specifies how DOM will receive control. LINKAGE=SVC specifies that linkage is by supervisor call, and LINKAGE=BRANCH specifies that linkage is by branch and link. LINKAGE=SVC is the default.

Use LINKAGE=BRANCH when you cannot issue an SVC.

Return and reason codes

Register 15 contains the following hexadecimal return codes from DOM LINKAGE=BRANCH:

Table 56. Return Codes for the DOM LINKAGE=BRANCH Macro

Hexadecimal Return Code	Meaning
00	**Meaning:** For LINKAGE=BRANCH, processing completed successfully. The system has accepted the request and will perform the deletion later.
40	**Meaning:** You issued DOM with LINKAGE=BRANCH, but the system could not obtain necessary storage. No processing was done.
4C	**Meaning:** You issued DOM with LINKAGE=BRANCH, but the system was unable to obtain storage for processing. No processing was done. This return code can occur as a result of unexpected error conditions in system storage.

There are no return codes from DOM LINKAGE=SVC.

Example 1

Issue a DOM by ID. The ID is in register 2.

```
R2     EQU     2

         DOM MSG=(R2)
```

Example 2

Issue a DOM by token.

```
         DOM TOKEN=READER
READER   DC    F'00000320'
```

Example 3

Issue a DOM by a list of IDs. The count is specified in a register. Use the branch-entry form of the macro.

```
R7     EQU     7
```

```
         L     R7,CURCOUNT              NUMBER OF ENTRIES IN LIST
         DOM MSGLIST=MYLIST,COUNT=(R7),SCOPE=SYSTEMS,LINKAGE=BRANCH
CURCOUNT DS    F
MYLIST   DS    60F
```

Chapter 47. DSPSERV — Create, delete, and control data spaces

DSPSERV for hiperspaces

To control the use of hiperspaces, use the variation of the DSPSERV macro described under Chapter 48, "DSPSERV — Create, delete, and control hiperspaces," on page 573.

Description

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. Unlike an address space, a data space can hold only data or programs stored as data. For more information on data spaces and how to use them, see *z/OS MVS Programming: Extended Addressability Guide*.

Use the DSPSERV macro to:

- Create a data space (CREATE parameter and TYPE=BASIC parameter)
- Delete a data space (DELETE parameter)
- Release an area of a data space (RELEASE parameter)
- Increase the current size of a data space (EXTEND parameter)
- Load an area of a data space into central storage (LOAD parameter)
- Take (that is, page out) from central storage an area of a data space (OUT parameter)
- Back data space virtual pages with 1 MB page frames, if possible (PAGEFRAMESIZE=1M).

DSPSERV is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the DREF, SCOPE, KEY, CALLERKEY, FPROT, and DISABLED parameters. These parameters are restricted to supervisor state or PSW key 0-7 programs.

Environment

The requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	To request the following DSPSERV services, a program *must* be supervisor state or PSW key 0-7: • Create a data space with disabled referenced (DREF) storage or fetch protected (FPROT) storage • Create and delete a SCOPE=ALL and SCOPE=COMMON data space • Assign a storage key to a data space • Load an area of a SCOPE=ALL or SCOPE=COMMON data space into central storage • Page out of central storage an area of a SCOPE=ALL or SCOPE=COMMON data space • Extend the current size of a data space it does not own

Environmental factor	Requirement
	Problem state programs with PSW key 8-F can request all other DSPSERV services for data spaces.
Dispatchable unit mode:	Task or SRB
Cross memory mode:	Any PASN, any HASN, any SASN
AMODE:	31- or 64-bit
ASC mode:	Primary or access register (AR)
Interrupt status:	Enabled for I/O and external interrupts, with the following exception: can be disabled for I/O and external interrupts if the caller specifies DSPSERV RELEASE with DISABLED=YES to release data space pages that reside in DREF storage
Locks:	No locks held, except the CPU lock if the caller specifies DSPSERV RELEASE with DISABLED=YES to release data space pages that reside in DREF storage
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 the caller is disabled and specifies DISABLED=YES, the parameter list must be in fixed or DREF storage.

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).

For information about programs in 64-bit addressing mode (AMODE 64), see *z/OS MVS Programming: Extended Addressability Guide*.

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 if the return code in GPR 15 is not 0; otherwise, used as a work register by the system

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the 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, BLOCKS, DREF, SCOPE, CALLERKEY, KEY, FPROT, TTOKEN, ORIGIN, NUMBLKS,BACK, PAGEFRAMESIZE
RELEASE	STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE, DISABLED
DELETE	STOKEN, TTOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
LOAD	STOKEN, START, BLOCKS
OUT	STOKEN, START, BLOCKS
PAGEFRAMESIZE=4K	**Default:** PAGEFRAMESIZE=4K

Syntax	Description
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	
,OUTNAME=*outname-addr*	*outname-addr*: RX-type address or register (2) - (12).
,START=*start-addr*	*start-addr*: RX-type address or register (2) - (12).
,BACK=31	**Default:** BACK=31 (for PAGEFRAMESIZE=4K)
,BACK=64	**Default:** BACK=64 (for PAGEFRAMESIZE=1M)
,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.
,DREF=NO	**Default:** DREF=NO
,DREF=YES	
,SCOPE=SINGLE	**Default:** SCOPE=SINGLE
,SCOPE=ALL	
,SCOPE=COMMON	

Syntax	Description
,CALLERKEY	**Default**: CALLERKEY
,KEY=*key-addr*	*key-addr*: RX-type address or register (2) - (12).
,FPROT=YES	**Default:** FPROT=YES
,FPROT=NO	
,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). **Default:** NUMRANGE=1
,VAR=NO	**Default:** VAR=NO
,VAR=YES	
,DISABLED=NO	**Default:** DISABLED=NO
,DISABLED=YES	
,PLISTVER=IMPLIED_VERSION	**Default:** IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 0
,HIDEZERO=NO	**Default:** HIDEZERO=NO
,HIDEZERO=YES	
,MF=S	

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.

Parameters

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, OUTNAME, GENNAME, BLOCKS, DREF, SCOPE, CALLERKEY, KEY, FPROT, TTOKEN, ORIGIN, NUMBLKS, BACK and PAGEFRAMESIZE.

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 supervisor state or key 0-7 program can release any data space it owns or created, if its home or primary address space is the same as the owner's. A problem state program can release any data space it owns or created.

The caller must own the data space, 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. 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.

If your program is disabled for I/O and external interrupts, use DISABLED=YES; otherwise, use DISABLED=NO (the default). DSPSERV RELEASE with DISABLED=YES is valid only to release data space pages that reside in DREF storage.

Use DSPSERV RELEASE instead of using the MVCL instruction for these reasons:

- DSPSERV RELEASE is faster than MVCL for very large areas.
- Pages that are released through DSPSERV RELEASE do not occupy space in real or auxiliary storage.

DELETE
Requests that the system delete a data space. STOKEN is the only required parameter on the DELETE request. TTOKEN is optional.

A problem state or key 8-F program can delete any data space it owns, providing its PSW key matches the storage key of the data space.

A supervisor state or key 0-7 program can delete any data space it owns or created, if its home or primary address space is the same as the owner's.

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 data space 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.

If the caller is problem state with PSW key 8 through F, any TCB can extend a data space that was created by any other TCB within the same address space.

If the caller is in supervisor state with PSW key 0 through 7, the TCB that represents the caller can be in any 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 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.

OUT
Tells the system that it can page some areas of a data space out of central storage. When you specify OUT, you must also specify the STOKEN, START, and BLOCKS parameters.

PAGEFRAMESIZE=4K
Backs data space virtual pages with 4 KB page frames at first reference.

PAGEFRAMESIZE=1M
Backs data space virtual pages with pageable 1 MB page frames at first reference. If pageable 1 MB page frames are not available at first reference, 4K page frames are used. If DEFINE IOON is later performed against pages backed with pageable 1 MB page frames, the pages are always backed above 2 GB. PAGEFRAMESIZE=1M is only valid for TYPE=BASIC data spaces. Refer to the BACK=64 and TYPE parameters for additional information.

,BACK=31
,BACK=64
Specifies the backing attributes of data space pages when defined as IOON (fixed).
Specifying ,BACK=31 backs the data space pages with frames that reside below 2 gigabytes when defined IOON. Specifying ,BACK=64 backs the data space pages by frames that reside either above or below 2 gigabytes when defined as IOON. Data spaces that are created by specifying PAGEFRAMESIZE=1M can only be backed by frames that reside above 2 gigabytes when defined as IOON. If ,BACK=31 is specified with PAGEFRAMESIZE=1M, the ,BACK=31 specification is ignored and the MNOTE ASMA254I message is generated, with the text PAGEFRAMESIZE=1M CANNOT BE SPECIFIED WITH BACK=31.

,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 services.

,TYPE=BASIC
Specifies that the system should create a data space rather than a hiperspace. TYPE=BASIC is the default.

,NAME=*name-addr*
Specifies the address of the eight-byte variable or constant that contains the name of the data space. NAME is required for DSPSERV CREATE.

Data space names are from one to eight bytes long. They can contain letters, numbers, and @, #, and $, but they cannot contain embedded blanks. Names that contain fewer than eight bytes must be left-justified and padded on the right with blanks.

Data space and hiperspace names must be unique within the home address space of the owner. No other data space or hiperspace in the home address space can have the same name. Therefore, in choosing names for your data spaces, you *must* avoid using the same names that IBM uses for data spaces. IBM uses the following names for data spaces and hiperspaces:

- Names that begin with A through I.
- Names that begin with SYSAxxxx through SYSIxxxx.

- Names that begin with numbers or the characters SYSDS.

Use the following names for your data spaces:

- Problem state programs can 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.
- Supervisor state programs and programs with PSW key 0 - 7 can use data space names that begin with @, #, $, or the letters J through Z. In addition, they can use names that begin with SYSJ through SYSZ. The system abends programs that begin names with SYSDS.

  Use names that begin with SYSJ through SYSZ to ensure that the names of the data spaces that belong to supervisor state programs and programs with PSW key 0 - 7 do not conflict with the names of data spaces that belong to problem state programs.

To ensure that the names for your data spaces 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 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 '*n*CCCCXYZ' 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 data space name it generated 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 data space. The address must be on a four-kilobyte boundary. START is required on RELEASE, LOAD, and OUT 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 or the size of an area within the data space.

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 then recompile the program to run on later releases of MVS, you must change the specification to BLOCKS=*((register))* before you recompile.

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 default is BLOCKS=0, setting the initial size and the maximum size equal to the installation (or IBM) default.

For a RELEASE request, BLOCKS is a required parameter that defines contiguous storage (in blocks of 4K bytes) that the system is to release (*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 of central storage.

For CREATE and EXPAND requests, for data spaces created with PAGEFRAMESIZE=1M where the BLOCKS specification does not result in the current size of the data space being a multiple of 256 blocks, the current partial last segment of the data space is backed by 4K pages at first reference. This ensures that the user cannot access 4K pages beyond the current end of the data space.

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 then recompile the program to run on later releases of MVS, you must change the specification to BLOCKS=*((register))* before you recompile.

,DREF=NO
,DREF=YES

Specifies whether (YES) or not (NO) disabled programs can reference the data space. If you specify NO, only enabled programs can reference the data space. If a disabled program references the data space, the system might abend the program. If you specify YES, both an enabled and a disabled program can reference the data space.

DREF is an optional parameter when you create a data space. The default, DREF=NO, specifies that only enabled programs can reference the data space.

,SCOPE=SINGLE
,SCOPE=ALL
,SCOPE=COMMON

Specifies whether the data space is a SCOPE=SINGLE, SCOPE=ALL, or a SCOPE=COMMON data space. A SCOPE=SINGLE data space may be referenced only by the owning address space. SCOPE=ALL and SCOPE=COMMON data spaces can be referenced by programs in many address spaces.

Any program can create and delete SCOPE=SINGLE data spaces. Only supervisor state or PSW key 0-7 programs can create and delete SCOPE=ALL and SCOPE=COMMON data spaces.

The address space that contains an owner of a SCOPE=ALL or SCOPE=COMMON data space must be nonswappable.

SCOPE is an optional parameter for DSPSERV CREATE; the default is SCOPE=SINGLE.

SCOPE=COMMON cannot be specified with a key (CALLERKEY, or KEY=) of 8-15.

,CALLERKEY
,KE=*key-addr*

Specifies the address of the eight-bit variable or constant that contains the storage key of the data space to be created. The key must be in bits 0-3 of the field. The system ignores bits 4-7. CALLERKEY specifies that the data space have the storage key that matches the PSW key of the caller.

The KEY parameter is optional on DSPSERV CREATE. CALLERKEY is the default.

A user key (8-15) cannot be specified for a SCOPE=COMMON data space.

,FPROT=YES
,FPROT=NO

Specifies whether the data space should (YES) or should not (NO) be fetch-protected. If you specify YES, the entire data space is fetch-protected. Fetch protection means a program must be in the key of the data space storage (or key 0) to reference data in the data space.

FPROT is an optional parameter for DSPSERV CREATE. The default, FPROT=YES, specifies that the data space is fetch-protected.

,TTOKEN=*ttoken-addr*

Specifies the address of the TTOKEN, the 16-byte variable or constant that identifies the TCB that is (for the CREATE request) to become the owner of the data space or is (for the DELETE request) the owner of the data space. Use this parameter when you assign ownership of a data space or when you delete a data space that belongs to another task. A program can assign ownership of a data space only when it creates it.

Before a program creates a data space and assigns ownership, it must know the TTOKEN of the TCB that is to be the new owner. The new owner must reside in the caller's home or primary address space.

If you do not specify TTOKEN, the system assumes the caller is to be the owner of the data space.

A problem state program with PSW key 8 - F can use the TTOKEN parameter only on the CREATE request and only to assign ownership to its own task or its job step task.

An SRB cannot own a data space. It can create one, but it must assign the data space to a TCB. The system abends SRB mode callers if they do not include the TTOKEN parameter on create requests.

,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*.

,VAR=YES
,VAR=NO

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 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 data space was created
- Rejects the request if the data space 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 data space.

,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 an input fullword that contains the address of the range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 8 bytes long. A mapping of each entry is provided through the mapping macro IARDRL. If you specify DISABLED=YES or a NUMRANGE value greater that 16, the range list must be in fixed storage.

,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, supplied through the RANGLIST parameter. For unauthorized callers, the maximum value is 16. The default is 1. If you specify INLIST=YES, you must specify RANGLIST.

,DISABLED=NO
,DISABLED=YES

Specifies that the caller is enabled for I/O and external interrupts (DISABLED=NO) or disabled for these interrupts (DISABLED=YES). DISABLED=NO is the default.

DISABLED=YES is valid only with DSPSERV RELEASE to release data space pages that reside in DREF storage. If you issue RELEASE and DISABLED=YES for a non-DREF data space, you receive an abend X'01D' with reason code X'020B'.

,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**, supports all parameters except those specifically referenced in higher versions.
- **1**, supports all parameters from version 0 and the BLOCKS(Positional_2) parameter.

To code, specify in this input parameter one of the following:

- IMPLIED_VERSION
- MAX
- A decimal value of 0-1.

,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 are shown in the following table.

Table 57. Return and Reason Codes for the DSPSERV CREATE Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: DSPSERV CREATE completed successfully. **Action**: None.

Table 57. Return and Reason Codes for the DSPSERV CREATE Macro (continued)

Hexadecimal Return Code	Hexadecimal 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 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.
08	xx0012xx	**Meaning**: Environmental error. The system's set of generated names for data spaces and hiperspaces has been temporarily exhausted. **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 are no longer exhausted. See also the description of the MAXCAD parameter in the IEASYSxx parmlib member in *z/OS MVS Initialization and Tuning Reference*.
0C	xx0007xx	**Meaning**: System error. The system cannot obtain addressability to its data structures. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

Hexadecimal return and reason codes from DSPSERV EXTEND are shown in the following table.

Table 58. Return and Reason Codes for the DSPSERV EXTEND Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**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                 10 MILLION BYTES OF STORAGE
DSPBLCKS DC   A((DSPCSIZE+4095)/4096)  NUMBER OF BLOCKS NEEDED FOR
*                                      A 10 MILLION BYTE DATA SPACE
```

Example 2

Release 9 ranges of storage in a data space with a previously built range list.

```
         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	**Default:** IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 0
,MF=(L,*list addr)*	*list addr*: Symbol.

Syntax	Description
,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, DREF, SCOPE, CALLERKEY, KEY, FPROT, TTOKEN, ORIGIN, NUMBLKS, BACK, PAGEFRAMESIZE
RELEASE	STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE, DISABLED
DELETE	STOKEN, TTOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
LOAD	STOKEN, START, BLOCKS
OUT	STOKEN, START, BLOCKS

Syntax	Description
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,TYPE=BASIC	**Default:** TYPE=BASIC
,NAME=*name-addr*	*name-addr*: RX-type address or register (2) - (12).
,GENNAME=NO	**Default:** GENNAME=NO
,GENNAME=COND	
,GENNAME=YES	
,OUTNAME=*outname-addr*	*outname-addr*: RX-type address or register (2) - (12).
PAGEFRAMESIZE=4K	**Default**: PAGEFRAMESIZE=4K
PAGEFRAMESIZE=1M	
,START=*start-addr*	*start-addr*: RX-type address or register (2) - (12).
,BACK=31	**Default**: BACK=31 (for PAGEFRAMESIZE=4K)
,BACK=64	**Default**: BACK=64 (for PAGEFRAMESIZE=1M)
,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.
,DREF=NO	**Default:** DREF=NO
,DREF=YES	
,SCOPE=SINGLE	**Default:** SCOPE=SINGLE
,SCOPE=ALL	
,SCOPE=COMMON	

Syntax	Description
,CALLERKEY	**Default**: CALLERKEY
,KEY=*key-addr*	*key-addr*: RX-type address or register (2) - (12).
,FPROT=YES	**Default:** FPROT=YES
,FPROT=NO	
,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). **Default:** NUMRANGE=1
,VAR=NO	**Default:** VAR=NO
,VAR=YES	
,DISABLED=NO	**Default:** DISABLED=NO
,DISABLED=YES	
,PLISTVER=IMPLIED_VERSION	**Default:** IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 0
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,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*)
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 48. DSPSERV — Create, delete, and control hiperspaces

DSPSERV for data spaces

To control the use of data spaces, use the variation of the DSPSERV macro described under Chapter 47, "DSPSERV — Create, delete, and control data spaces," on page 555.

Description

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. A hiperspace can hold user data and programs stored as data. Data is not directly addressable; to manipulate data in a hiperspace, you use the HSPSERV macro to bring the data into the address space in blocks of 4K bytes.

Supervisor state or PSW key 0 through 7 programs have a choice of creating a standard hiperspace or an ESO hiperspace. The **standard hiperspace** is backed with real storage and auxiliary storage, if necessary. The HSTYPE=SCROLL parameter creates a standard hiperspace. The **ESO hiperspace** is backed only with real storage. HSTYPE=CACHE creates an ESO hiperspace. For more information on hiperspaces and how to use them, see *z/OS MVS Programming: Extended Addressability Guide*. To learn the restrictions for the use of hiperspaces, see the description of the HSPSERV macro.

Use the DSPSERV macro to:

- Create a 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)

DSPSERV is also described in *z/OS MVS Programming: Assembler Services Reference ABE-HSP*, with the exception of the KEY, CALLERKEY, TTOKEN, HSTYPE, SHARE, DISABLED, and CASTOUT parameters. These parameters are restricted to supervisor state or PSW key 0-7 programs.

Environment

Requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	To request the following DSPSERV services, a program *must* be supervisor state or PSW key 0-7: • Create and delete an ESO or a shared standard hiperspace • Release storage in a shared or ESO hiperspace • Extend the current size of a shared or ESO hiperspace • Assign a storage key to a hiperspace • Assign hiperspace ownership to a TCB Problem state programs with PSW key 8-F can request all other DSPSERV services.
Dispatchable unit mode:	Task or SRB

Environmental factor	Requirement
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, with the following exception: can be disabled for I/O and external interrupts if the caller specifies DSPSERV RELEASE with DISABLED=YES to release an ESO (HSTYPE=CACHE) hiperspace
Locks:	No locks held, except the CPU lock if the caller specifies DSPSERV RELEASE with DISABLED=YES to release an ESO (HSTYPE=CACHE) hiperspace
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 the caller is disabled and specifies DISABLED=YES, the parameter list must be in fixed or disabled reference (DREF) storage.

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).

For information about programs in 64-bit addressing mode (AMODE 64), see *z/OS MVS Programming: Extended Addressability Guide*.

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 if the return code in GPR 15 is not 0; otherwise, used as a work register by the system

1
Used as a work register by the system

2-13
Unchanged

14
Used as a work register by the system

15
Return code

When control returns to the caller, the 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, HSTYPE, CASTOUT, SHARE GENNAME, OUTNAME, BLOCKS, CALLERKEY, KEY, FPROT, TTOKEN, ORIGIN, and NUMBLKS
RELEASE	STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE, DISABLED
DELETE	STOKEN, TTOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,TYPE=HIPERSPACE	
,HSTYPE=SCROLL	**Default:** HSTYPE=SCROLL
,HSTYPE=CACHE	

Syntax	Description
,SHARE=NO	**Default:** SHARE=NO
,SHARE=YES	**Note:** SHARE is valid only if you specify HSTYPE=SCROLL.
,CASTOUT=YES	**Default:** CASTOUT=YES
,CASTOUT=NO	**Note:** CASTOUT is valid only if you specify HSTYPE=CACHE.
,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.
,CALLERKEY	**Default**: CALLERKEY
,KEY=*key-addr*	*key-addr*: RX-type address or register (2) - (12).
,FPROT=YES	**Default:** FPROT=YES
,FPROT=NO	
,TTOKEN=*ttoken-addr*	*ttoken-addr*: RX-type address or register (2) - (12).
,ORIGIN=*origin-addr*	*origin-addr*: RX-type address or register (2) - (12).

Syntax	Description
,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). **Default:** NUMRANGE=1
,VAR=NO	**Default:** VAR=NO
,VAR=YES	
,DISABLED=NO	**Default:** DISABLED=NO
,DISABLED=YES	**Note:** DISABLED=YES is valid only if you specify DSPSERV RELEASE with HSTYPE=CACHE.
,PLISTVER=IMPLIED_VERSION	**Default:** IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 0
,MF=S	

The CREATE, RELEASE, DELETE, and EXTEND parameters, which designate the services of the DSPSERV macro, are mutually exclusive. You can select only one.

Parameters

The parameters are explained as follows:

CREATE

Requests that the system create a 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 the NAME, TYPE=HIPERSPACE, and STOKEN parameters. To create an ESO or a shared standard hiperspace, your program must be supervisor state or have PSW key 0 - 7.

Optional parameters when you create a hiperspace are: HSTYPE, CASTOUT, GENNAME, OUTNAME, BLOCKS, KEY, CALLERKEY, FPROT, TTOKEN, ORIGIN, SHARE, 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.

A problem state or PSW key 8 - F caller must own the hiperspace, and its PSW key must be zero or equal to the key of the storage the system is to release. A supervisor state or PSW key 0 - 7 caller must have its home or primary address space the same as the owner's home address space, and its PSW key must be zero or equal to the key of the storage the system is to release.

If your program is disabled for I/O and external interrupts, use DISABLED=YES; otherwise, use DISABLED=NO (the default). DSPSERV RELEASE with DISABLED=YES is valid only to release an ESO (HSTYPE=CACHE) hiperspace.

Pages that are released through DSPSERV RELEASE do not occupy space in real 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. TTOKEN is optional.

A problem state or key 8-F program can delete any hiperspace it owns and for which its PSW key matches the key of the hiperspace.

A supervisor state or key 0-7 program can delete any hiperspace it owns and other hiperspaces, if its home or primary address space is the same as the owner's.

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.

If the caller is problem state and PSW key 8 through F, it must own the hiperspace. Otherwise, the TCB that represents the caller must be in the home or primary address of the owner of 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. DSPSERV CREATE returns the STOKEN as output. STOKEN is required input for all other DSPSERV requests.

,TYPE=HIPERSPACE
Specifies that the system is to create a hiperspace rather than a data space.

,HSTYPE=SCROLL
,HSTYPE=CACHE
Specifies the type of hiperspace the system is to create: HSTYPE=SCROLL creates a standard hiperspace, the type of storage area that your program can scroll through. HSTYPE=CACHE creates an ESO hiperspace, one that acts as a high-speed cache for storing data. HSTYPE=SCROLL is the default.

,SHARE=NO
,SHARE=YES

Specifies whether the system is to create a nonshared standard hiperspace (SHARE=NO) or a shared standard hiperspace (SHARE=YES). This parameter is valid only if you specify HSTYPE=SCROLL. When you specify HSTYPE=SCROLL, SHARE=NO is the default.

Generally, a program can share a **nonshared standard** hiperspace only with programs that are dispatched in the owner's home address space. However, a program not dispatched in the owner's home address space and using an ALET, can access this nonshared standard hiperspace through the owner's home PASN-AL. A program can share a **shared standard** hiperspace with programs that are dispatched in any address space.

,CASTOUT=YES
,CASTOUT=NO

Specifies that the system is to persist (CASTOUT=NO) or not persist (CASTOUT=YES) in retaining a copy of the data in the hiperspace. The CASTOUT parameter is valid only if you specify HSTYPE=CACHE. When you specify HSTYPE=CACHE, CASTOUT=YES is the default.

When the system needs the real storage for its own needs, it is less likely to take the real storage from a hiperspace created with CASTOUT=NO than from one created with CASTOUT=YES.

CASTOUT=YES indicates that the system can discard the data when it needs the real storage for other purposes. CASTOUT=NO specifies that the system is to give the data in the ESO hiperspace more priority when searching for pages to remove from real storage when a shortage arises.

Note: Specifying CASTOUT=NO places a heavy demand on real storage. The system might discard the pages regardless of CASTOUT=NO. For example, if the system swaps out the address space that owns the hiperspace, it discards pages without regard to CASTOUT. (To prevent the loss due to a swapped-out address space, make the address space that owns the hiperspace nonswappable.)

,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.

Names of hiperspaces and data spaces 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 uses for data spaces and hiperspaces. IBM uses the following names:

- Names that begin with A through I.
- Names that begin with SYSAxxxx through SYSIxxxx.
- Names that begin with numbers or the characters SYSDS.

Use the following names for your hiperspaces:

- Problem state programs can 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.
- Supervisor state programs and programs with PSW key 0 - 7 can use hiperspace names that begin with @, #, $, or the letters I through Z. In addition, they can use names that begin with SYSJ through SYSZ. The system abends programs that begin names with SYSDS.

  Use names that begin with SYSJ through SYSZ to ensure that the names of the hiperspaces that belong to supervisor state programs and programs with PSW key 0 - 7 do not conflict with the names of hiperspaces that belong to problem state programs.

To ensure that the names for your hiperspaces 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 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 (or the whole name if it has three or fewer characters). For example, if you supply 'XYZDATA' on the NAME parameter, the name becomes '*n*CCCCXYZ' 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 keyword 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 generates for 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 address of a four-byte variable that contains the size of the hiperspace or the size of an area within the hiperspace.

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 default is 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 to the current size of the hiperspace.

,CALLERKEY
,KEY=*key-addr*
Specifies the address of the eight-bit variable or constant that contains the storage key of the hiperspace to be created. The key must be in bits 0-3 of the field. The system ignores bits 4-7. CALLERKEY specifies that the hiperspace is to have the storage key that matches the PSW key of the caller.

The KEY parameter is optional on DSPSERV CREATE. CALLERKEY is the default.

,FPROT=YES
,FPROT=NO
Specifies whether the hiperspace should (YES) or should not (NO) be fetch-protected. If you specify YES, the entire hiperspace is fetch-protected. Fetch protection means a program must be in the key of the hiperspace storage (or key 0) to reference data in the hiperspace.

FPROT is an optional parameter for DSPSERV CREATE. The default, FPROT=YES, specifies that the hiperspace is fetch-protected.

,TTOKEN=*ttoken-addr*
Specifies the address of the TTOKEN, the 16-byte variable or constant that identifies the TCB that is (for the CREATE request) to become the owner of the hiperspace or is (for the DELETE request) the owner of the hiperspace. Use this parameter when you assign ownership of a hiperspace or when you delete a hiperspace that belongs to another task. A program can assign ownership of a hiperspace only when it creates it.

Before a program creates a hiperspace and assigns ownership, it must know the TTOKEN of the TCB that is to be the new owner. The new owner must reside in the caller's home or primary address space.

If you do not specify TTOKEN, the system assumes the caller is the owner.

An SRB cannot own a hiperspace. A program that the SRB represents can create one, but it must assign the hiperspace to a TCB. The system abends SRB mode callers if they do not include the TTOKEN parameter on create requests.

,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.

,VAR=YES
,VAR=NO
Specifies whether or not 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 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 is unable to satisfy the 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.

,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 an input fullword that contains the address of the range list. The range list consists of a number of entries (as specified by NUMRANGE) where each entry is 8 bytes long. A mapping of each entry is provided through the mapping macro IARDRL. If you specify DISABLED=YES or a NUMRANGE value greater that 16, the range list must be in fixed storage.

,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, supplied through the RANGLIST parameter. For unauthorized callers, the maximum value is 16. The default is 1. If you specify INLIST=YES, you must specify RANGLIST.

,DISABLED=NO
,DISABLED=YES
Specifies that the caller is enabled for I/O and external interrupts (DISABLED=NO) or disabled for these interrupts (DISABLED=YES). DISABLED=NO is the default.

DISABLED=YES is valid only with DSPSERV RELEASE to release an ESO (HSTYPE=CACHE) hiperspace. If you issue RELEASE and DISABLED=YES for a standard (HSTYPE=SCROLL) hiperspace, you receive an abend X'01D' with reason code X'020B'.

,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 more information.

Return and reason codes

Hexadecimal return and reason codes from DSPSERV CREATE are shown in the following table.

Table 59. Return and Reason Codes for the DSPSERV CREATE Macro

Hexadecimal Return Code	Hexadecimal Reason Code	Meaning and Action
00	None	**Meaning**: DSPSERV CREATE completed successfully. **Action**: None.
04	xx000Cxx	**Meaning**: Program error. DSPSERV CREATE completed successfully. You specified a size of 2-gigabytes (524,288 blocks). However, because the processor did not support a hiperspace with zero origin, a hiperspace of one less block (524,287 blocks) was created. **Action**: None required. However, you should verify that your program correctly accounts for the nonzero origin of the hiperspace.
08	xx0005xx	**Meaning**: Program error. Creation of the hiperspace would violate installation criteria. See the IEFUSI installation exit in *z/OS MVS Installation Exits*. **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.

Table 59. Return and Reason Codes for the DSPSERV CREATE Macro (continued)		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
08	xx0010xx	**Meaning**: Environmental error. ESO hiperspace creation was rejected because there is no real storage on the system. **Action**: Determine if an ESO hiperspace is required. If not, modify the program to specify a standard rather than an ESO hiperspace. If an ESO hiperspace is required, run the program on another system with real storage installed.
08	xx0012xx	**Meaning**: Environmental error. The system's set of generated names for data spaces and hiperspaces has been temporarily exhausted. **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 are no longer exhausted.
0C	xx0006xx	**Meaning**: Environmental error. The system cannot create any additional hiperspaces 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 are no longer exhausted.
0C	xx0007xx	**Meaning**: System error. The system cannot obtain addressability to its own hiperspaces. **Action**: Record the return and reason codes and supply them to the appropriate IBM support personnel.

Hexadecimal return and reason codes from DSPSERV EXTEND are shown in the following table.

Table 60. Return and Reason Codes for the DSPSERV EXTEND Macro		
Hexadecimal Return Code	**Hexadecimal Reason Code**	**Meaning and Action**
00	None	**Meaning**: DSPSERV EXTEND completed successfully. **Action**: None.
08	xx0502xx	**Meaning**: Environmental error. Extending the hiperspace size would cause the data space and hiperspace limits for the address space to be exceeded. **Action**: Check with your system programmer, who might be able to tune the system so that the function is made available to your program.
08	xx0503xx	**Meaning**: Program error. You are using VAR=YES to extend the current size of the hiperspace; however, the hiperspace is already the maximum size. **Action**: None required. However, if your program requires more storage, you should consider creating an additional hiperspace.

The caller of DSPSERV does not receive any return codes for the RELEASE and DELETE services.

Example 1

Create a hiperspace named TEMP with a size of 10 million bytes.

```
DSP1     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
```

Example 2

Release 9 ranges of storage in a data space with a previously built range list.

```
         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	**Default:** IMPLIED_VERSION
,PLISTVER=MAX	
,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, HSTYPE, SHARE, CASTOUT, GENNAME, OUTNAME, BLOCKS, CALLERKEY, KEY, FPROT, TTOKEN, ORIGIN, and NUMBLKS
RELEASE	STOKEN, START, BLOCKS, INLIST, RANGLIST, NUMRANGE, DISABLED
DELETE	STOKEN, TTOKEN
EXTEND	STOKEN, BLOCKS, VAR, NUMBLKS
,STOKEN=*stoken-addr*	*stoken-addr*: RX-type address or register (2) - (12).
,TYPE=HIPERSPACE	
,HSTYPE=SCROLL	**Default:** HSTYPE=SCROLL
,HSTYPE=CACHE	

Syntax	Description
,SHARE=NO	**Default:** SHARE=NO
,SHARE=YES	**Note:** SHARE is valid only if you specify HSTYPE=SCROLL.
,CASTOUT=YES	**Default:** CASTOUT=YES
,CASTOUT=NO	**Note:** CASTOUT is valid only if you specify HSTYPE=CACHE.
,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.
,KEY=*key-addr*	*key-addr*: RX-type address or register (2) - (12).
,CALLERKEY	**Default**: CALLERKEY
,FPROT=YES	**Default:** FPROT=YES
,FPROT=NO	
,TTOKEN=*ttoken-addr*	*ttoken-addr*: RX-type address or register (2) - (12).
,ORIGIN=*origin-addr*	*origin-addr*: RX-type address or register (2) - (12).

Syntax	Description
,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). **Default:** NUMRANGE=1
,VAR=NO	**Default:** VAR=NO
,VAR=YES	
,DISABLED=NO	**Default:** DISABLED=NO
,DISABLED=YES	**Note:** DISABLED=YES is valid only if you specify DSPSERV RELEASE with HSTYPE=CACHE.
,PLISTVER=IMPLIED_VERSION	**Default:** IMPLIED_VERSION
,PLISTVER=MAX	
,PLISTVER=*plistver*	*plistver*: 0
,MF=(E,*list addr*)	*list addr*: RX-type address or register (2) - (12).
,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 49. DYNALLOC — Dynamic allocation

Description

Use the DYNALLOC macro to invoke dynamic allocation functions. Before attempting to use this macro, you must read the chapters “Dynamic Allocation” and “Requesting Dynamic Allocation Functions” in *z/OS MVS Programming: Authorized Assembler Services Guide*, for complete information on DYNALLOC.

Environment

Requirements for the caller are:

Environmental factor	Requirement
Minimum authorization:	Problem state or supervisor 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 requirement
Control parameters:	Must be in the primary address space

Programming requirements

The calling program must include the following mapping macros to construct the SVC 99 parameter list:

- IEFZB4D0
- IEFZB4D2

See *z/OS MVS Programming: Authorized Assembler Services Guide* for details on constructing the parameter list.

Restrictions

See *z/OS MVS Programming: Authorized Assembler Services Guide* for programming restrictions and limitations.

Register information

On entry to the macro, general purpose register 1 must contain the address of a pointer to the SVC 99 parameter list structure. See *z/OS MVS Programming: Authorized Assembler Services Guide* for a detailed description of the parameter list.

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

When control is returned to the calling program the GPRs contain:

Register
Contents

0-1
Used as work registers by the system

2-14
Unchanged

15
Return code

Performance implications

There are no performance implications when the restrictions and limitations are all met.

Syntax

The DYNALLOC macro is written as follows:

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

Parameters

There are no parameters for DYNALLOC.

Return and reason codes

When control returns from DYNALLOC, GPR 15 contains a return code. The return codes and associated reason codes are described in *z/OS MVS Programming: Authorized Assembler Services Guide*.

Appendix A. Accessibility

Accessible publications for this product are offered through IBM Documentation (www.ibm.com/docs/en/zos).

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 Documentation 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 IBM Documentation. 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 authorized 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

M

N

O

P

R

S

U

V

X

Product Number: 5650-ZOS

SA23-1372-40